11/16

2020년 11월 16일 월요일 오전 9:42

- SQL튜닝 수업
  데이터 분석가들에게 SQL 작성능력은 필수
  데이터 분석가들이 데이터 분석을 하는 회사들의 데이터가 대부분 대용량 데이터이기 때문에 데이터를 검색하는 속도가 많이 느리므로 빠른 데이터 분석을 위해서는 SQL튜닝 기술도 잘 알고 있어야 합니다.

- SQL튜닝이란?
  데이터 검색속도를 향상시키는 기술

■ 인덱스 엑세스 방법 8가지

	인덱스 엑세스 방법	힌트
1.	index range scan	index
2.	index unique scan	index
3.	index skip scan	index_ss
4.	index full scan	index_fs
5.	index fast full scan	index_ffs
6.	index merge scan	and_equal
7.	index bitmap merge scan	index_combine
8.	index join	index_join

- 오라클 힌트(hint) ?
  오라클 옵티마이저가 SQL을 수행할 때 실행 계획을 만드는데 실행계획을 SQL 사용자가 조정하는 명령어

- 실행계획은 누가 만드는가?
  옵티마이져라는 프로세서가 만듭니다.
  옵티마이져에게 문법에 맞는 적절한 힌트를 주면 옵티마이져는 사용자가 요청한대로 실행계획을 만듭니다.

- 실행계획 보는 방법:

1. emp와 dept테이블 재생성 스크립트를 돌립니다.
2. 아래의 SQL의 실행계획을 확인합니다.

```
explain plan for
select ename, sal
       from emp
       where sal = 1400;
select *
  from table(dbms_xplan.display);
```

```
-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    20 |     3   (0)| 00:00:01 |  (2)
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    20 |     3   (0)| 00:00:01 | (1)
-------------------------------------------------------------------------
```

※ 설명 : 실행계획에 full table scan으로 나오면 emp 테이블을 처음부터 끝까지 다 스캔했다는 뜻 입니다.
위의 실행계획을 보는 방법은 SQL의 결과는 못보고 그냥 단지 실행계획만 확인하는 방법입니다.

- SQL의 결과도 확인하면서 실행계획 확인하는 방법

```
select /*+ gather_plan_statistics */ ename, sal
       from emp
       where sal = 1300;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

이렇게 하면 아래의 결과가 나온다.

```
-----------------------------------------------------------------------------------
```

```
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------------
```

buffers의 갯수가 적을수록 좋다

※ 설명 : 맨 뒤에 나오는 buffer의 갯수가 줄어들수록 튜닝이 잘된 것입니다.

Ex 1) 위의 SQL을 튜닝하시오!

a. emp테이블의 sal에 인덱스를 생성하시오

```
create index emp_sal
    on emp(sal);
```

하고 다시 결과를 보기

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where sal = 1300;

SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

- **근데 속도가 안빨라졌다?**

```
select /*+ gather_plan_statistics index(emp emp_sal) */ ename, sal
      from emp
      where sal = 1300;
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_SAL |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

(3) (2) (1) 실행순서

※ 설명 : buffer의 갯수가 튜닝전 7개에서 튜닝 후 3개로 줄어든 것을 확인할 수 있습니다.

11/17 (인덱스 엑세스)

2020년 11월 17일 화요일 오전 9:51

SQLD : SQL개발자 (SQL수업)
SQLP: SQL 튜너(SQL수업 + SQL튜닝 수업)
ADSP : 데이터 분석가

- SQL튜닝 목차

1. 인덱스 SQL튜닝
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL튜닝

SQL table scan과 index scan 의 성능상 차이를 실행계획을 통해서 경험했습니다.

1. Index range scan
   인덱스를 range(부분) 하게 먼저 스캔하고 인덱스에서 얻은 rowid로 테이블의 데이터를 엑세스하는 스캔 방법

   Ex) 목차(인덱스) + 책(테이블)
   책의 페이지 번호 : rowid
   인덱스가 없다면 데이터를 검색하기 위해서 full table scan을 할 수 밖에 없지만 인덱스가 있다면 인덱스(목차)를 통해서 먼저 데이터를 검색하고 인덱스의 rowid를 통해서 테이블을 엑세스 할 수 있게 됩니다.

   Ex) select ename, sal
   from emp
   where ename = 'BLAKE';
   Δ
   create index emp_ename
   on emp(ename);

   ```
   select /*+ gather_plan_statistics */ ename, sal
   from emp
   where ename = 'BLAKE';
   SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
   ```

   ```
   ---------------------------------------------------------------------------------
   | Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
   ---------------------------------------------------------------------------------
   |   0 | SELECT STATEMENT  |      |      1 |        |      1 |00:00:00.01 |       7 |
   |* 1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
   ---------------------------------------------------------------------------------
   ```

   full table scan은 emp테이블을 처음부터 끝까지 다 스캔한 것입니다.

- 위의 SQL을 인덱스 스캔이 될 수 있도록 힌트를 주고 실행계획을 보시오

```
select /*+gather_plan_statistics index(emp emp_ename) */ename, sal
        from emp
        where ename = 'BLAKE';
```

index(emp emp_ename : emp 테이블의 emp_ename 인덱스를 실행해라

❖ 설명 : gather_plan_statistics 힌트? 실제 실행계획을 보는 힌트

실행계획 2가지? 1. 예상 실행계획 : SQL을 실행하기 전에 실행

```
explain plan for
    select ename, sal
        from emp
        where ename = 'BLAKE';
```

2. 실제 실행계획 : SQL을 직접 실행하면서 바로 나오는 그 실행계획

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

패키지 프로시져(인자값1, 인자값2, 인자값3)

'ALLSTATS LAST' : buffers를 보려면 반드시 필요

❖ 암시적 형변환 사용시 주의 사항

```
select ename, job, sal
        from emp
        order by sal desc;

select decode(job, 'PRESIDENT', null, sal)
        from emp;
```

➢ 인덱스 엑세스 방법 8가지

1. index range scan

- insex rance scan
  "non unique"한 인덱스를 엑세스하는 스캔 방법"
- 인덱스의 종류 : 1. unique 인덱스: 값이 중복되지 않고 unique해야 생성되는 인덱스
  2. non unique 인덱스 : 값이 중복되어도 상관 없이 그냥 생성되는 인덱스

unique 인덱스 생성 예제)

1. 데모 돌린다. (@demo.sql)
2. create unique index emp_empno on emp(empno); <-- 생성됨
   create unique index emp_deptno on emp(deptno); <--- 생성안됨
   >>> ORA-01452: 중복 키가 있습니다. 유일한 인덱스를 작성할 수 없습니다
   01452. 00000 - "cannot CREATE UNIQUE INDEX; duplicate keys found"
   이렇게 나옴

- index unique scan
  "unique 인덱스를 통해서 테이블을 엑세스하는 스캔 방법"
  unique 인덱스는 중복된 데이터가 없어야지만 인덱스가 걸린다.

Ex) create unique index emp_empno on emp(empno);

primary key제약이나 unique 제약을 컬럼에 걸면 unique인덱스가 자동으로 생성됩니다.

- 지금까지 만든 인덱스 리스트를 확인하시오!

```
select index_name, uniqueness
        from user_indexes
        where table_name = 'EMP';

EMP_EMPNO  UNIQUE
EMP_ENAME  NONUNIQUE
EMP_JOB    NONUNIQUE
```

이렇게 나옵니다.

```
select index_name, uniqueness
    from user_indexes
    where table_name = 'DEPT';

DEPT_DEPTNO_PK     UNIQUE 이렇게 나온다.


DEPT_DEPTNO_PK
```

❖ 설명 : 인덱스 이름이 제약이름으로 생성되었습니다.

- index full scan

"인덱스 전체를 full로 읽어서 원하는 데이터를 엑세스 하는 방법"

Ex) 사원 테이블의 인원수가 전부 몇명인가?

```
select */+ gather_plan_statistics */ count(*)
        from emp;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-------------------------------------------------------------------------------------
| Id  | Operation         | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |            |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT AGGREGATE   |            |      1 |      1 |      1 |00:00:00.01 |       1 |
|   2 |   INDEX FULL SCAN| EMP_EMPNO |      1 |     14 |     14 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------
```

인덱스를 써주면 오라클이 자체로

```
select */+ gather_plan_statistics */ count(empno)
        from emp;
```

로 바꿔줌. 그래서 인덱스 스캔 한 것임

근데 이렇게 안되고 전체를 스캔한다?

그럼

```
select /*+ gather_plan_statistics index_fs(emp emp_empno) */ count(empno)
        from emp;
```

해준다.

❖ 설명 : 사원수가 몇명인지 확인하는 쿼리의 성능을 높이기 위해서 emp_empno 인덱스를 full scan하여 사원수를 카운트 했습니다.

그리고 버퍼의 갯수는 1개 입니다.

- Index fasf full scan
  "덱스를 처음부터 끝까지 스캔하는 방법은 index full scan과 똑같으나 index fast full scan이 index full scan과 다른점은 인덱스를 full로 읽을 때 multi block i/o를 합니다."

  multi block i/o란 ? 책의 앞에 목차가 1페이지~ 100 페이지라고 한다면 index full scan은 한페이지 한페이지 하나씩 넘기는게 index full scan이고 index fast full scan은 한번 넘길때 10페이지씩 넘기는 것입니다.

  Ex) @demo.sql 하고
  ```
  create index emp_job on emp(job);
  직업, 직업별 인원수를 출력하시오!

  select /*+ gather_plan_statistics */ job, count(*)
    from emp
    group by job;
  SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
  ```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      5 |00:00:00.01 |       7 |       |       |          |
|   1 |  HASH GROUP BY     |      |      1 |     14 |      5 |00:00:00.01 |       7 |  1345K|  1345K|  753K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

  Full scan 하니까 힌트를 줘야한다.

```
select /*+ gather_plan_statistics index_ffs(emp emp_job) */ job, count(job)
  from emp
  group by job;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

  이렇게 해도 full scan된다.

❖ **중요설명 : 위와 같이 index_ffs 힌트를 줬어도 테이블 full scan을 하는 이유는 job컬럼에 not null 제약이 없어서 입니다. index fast full scan을 하려면 not null제약이 있어야 합니다.**

- job에 not null제약 생성하기
  ```
  alter table emp
    modify job constraint emp_job_nn not null;
  ```

  그리고 다시
  ```
  select /*+ gather_plan_statistics index_ffs(emp emp_job) */ job, count(job)
    from emp
    group by job;
  SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
  ```
  해주면

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |      1 |        |      5 |00:00:00.01 |       4 |       |       |          |
|   1 |  HASH GROUP BY     |         |      1 |     14 |      5 |00:00:00.01 |       4 |  1345K|  1345K|  772K (0)|
```

```
|  2 |  INDEX FAST FULL SCAN| EMP_JOB |      1 |     14 |     14 |00:00:00.01 |       4 |       |       |          |
--------------------------------------------------------------------------------------------------------------------------------------
```

이렇게 된다.

- Index full scan과 index fast full scan의 차이점?
  1. index full scan 보다는 index fast full scan이 더 빠릅니다.
  2. index full scan은 데이터 정렬이 보장이 되지만
     index fast full scan은 결과로 출력되는 데이터의 정렬이 보장되지 않습니다.
     정렬이 되야 하는 경우에는 full scan으로 해주는게 좋고 정렬이 안되도 되는 경우에는 fast full scan 해주는게 좋다.

11/18 (인덱스튜닝)

2020년 11월 18일 수요일 오전 9:34

■ SQL 목차

(확실하게 알고 있어야 할 5가지)

1. 인덱스 SQL 튜닝
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL튜닝

■ 3. index skip scan

1. 단일 컬럼 인덱스 : 컬럼을 하나로 해서 만든 인덱스
   Ex) create index emp_deptno
   on emp(deptno);
2. 결합 컬럼 인덱스 : 컬럼을 여러개로 해서 만든 인덱스 (현업에서 주로 사용)
   Ex) create index emp_deptno_sal
   on emp(deptno, sal);

- emp_deptno_sal 결합 컬럼 인덱스의 구조 (컬럼값 + rowid)

```
select deptno, sal, rowid
    from emp
    where deptno >=0;
```

설명 : deptno를 먼저 ascending 하게 정렬하고 그것을 기준으로 월급을 ascending하게 정렬되게끔 인덱스를 구성 하고 있습니다.

- 결합 컬럼 인덱스가 필요한 이유?

-->인덱스에서 데이터를 읽어오면서 테이블 엑세스를 줄일수 있기 때문입니다.

(목차) (책)

내가 검색하고자 하는 데이터가 목차에 있어서 책을 안 뒤지고 목차에서만 읽고 끝났다면 빠르게 데이터를 검색한 것입니다.

책보다는 목차가 훨씬 두께가 얇으므로 목차에서 원하는 데이터를 찾는게 책에서 찾는 것보다 훨씬 속도가 빠릅니다.

- 인덱스 목록 조회하기

```
select index_name
    from user_indexes
    where table_name = 'EMP';
```

- 단일 컬럼 인덱스를 통해서 테이블 엑세스

```
select /*+ gather_plan_statistics index(emp emp_deptno) */ deptno, sal
    from emp
    where deptno = 10;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |      3 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP        |      1 |      3 |      3 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN           | EMP_DEPTNO |      1 |      3 |      3 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

salary정보를 가져오기 위해서 Table Access를 한 것임.

select절의 sal을 빼다면 table access를 안 함.

```
-----------------------------------------------------------------------------------
| Id  | Operation         | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |            |      1 |        |      3 |00:00:00.01 |       1 |
|*  1 |  INDEX RANGE SCAN| EMP_DEPTNO |      1 |      3 |      3 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------
```

- 위의 쿼리를 결합 컬럼 인덱스를 통해서 조회 해본다.

```
select /*+ gather_plan_statistics index(emp emp_deptno_sal) */ deptno, sal
    from emp
    where deptno = 10;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------
| Id  | Operation         | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |      1 |        |      3 |00:00:00.01 |       1 |
|*  1 |  INDEX RANGE SCAN| EMP_DEPTNO_SAL |      1 |      3 |      3 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------
```

설명 : 버퍼의 개수가 2개에서 1개로 줄어들면서 성능이 2배 빨라졌습니다.

현업에서는 주로 단일 컬럼 인덱스보다는 결합 컬럼 인덱스가 많습니다.

■ Index skip scan

" 인덱스를 full scan또는 fast full scan으로 전체 스캔하는게 아니라 중간중간 skip해서 원하는 데이터를 검색하는 스캔 방법"

■ **인덱스 엑세스 방법 8가지**

인덱스 엑세스 방법	힌트
1. index range scan	index index_asc : index ascending 하게 스캔 index_desc: index descending 하게 스캔
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

검색할 필요가 없는 부분을 skip하는 scan방법

- emp_deptno_sal 인덱의 구조
  @demo.sql
  create index emp_dept_sal on emp(deptno, sal);
  select deptno, sal, rowid
  from emp
  where deptno >= 0;

※ 중요설명 : 쿼리문에서 결합 컬럼 인덱스를 사용하려면 쿼리문의 where절에 결합컬럼 인덱스의 첫번째 컬럼이 where절에 반드시 있어야 합니다.

emp_deptno_sal

1. emp_deptno_sal 결합 컬럼 인덱스의 첫번째 컬럼인 deptno가 where 절에 있을 경우 index range scan 으로 실행됩니다.

```
select /*+ gather_plan_statistics */ ename, deptno, sal
      from emp
      wher deptno = 20;
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |      1 |        |      5 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP          |      1 |      5 |      5 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN           | EMP_DEPT_SAL |      1 |      5 |      5 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

2. emp_deptno_sal결합 컬럼 인덱스의 두번째 컬럼이 where절에 있으면 실행계획이 index full scan 또는 table full scan으로 스캔합니다.

```
select /*+ gather_plan_statistics */ ename, deptno, sal
      from emp
      where sal = 3000;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      2 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      2 |      2 |00:00:00.01 |       7 |
-----------------------------------------------------------------------------------
```

설명 : 위와 같이 index range scan으로 출력되지 않고 index full scan으로 출력되고 있습니다.
그러면 성능이 좋지 않으므로 이를 해결하기 위한 방법이 index skip scan입니다.

여기에

```
select /*+ gather_plan_statistics */ ename, deptno, sal
      from emp
      where sal = 3000 and deptno > 0;
```

이렇게 하면 range scan을 타긴 타지만 데이터량이 많은 경우 느려질 수 있으므로 skip scan을 사용하도록 한다.

- Index skip scan의 원리

```
select deptno, sal, rowid
      from emp
      where deptno >= 0;

select /*+ gather_plan_statistics */ ename, deptno, sal
      from emp
      where sal = 2975;
```

----> range 스캔 불가능/ deptno는 range scan 가능

```
-----------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      1 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
-----------------------------------------------------------------------------------
```

설명 : deptno와는 다르게 sal은 emp_deptno_sal 인덱스에서 deptno처럼 정렬되어 있지 않기 때문에 월급이 2975를 조회하기 위해 인덱스를 처음부터 끝까지 다 full로 scan해야 합니다.
그러면서 불필요한 데이터까지 다 스캔을 해야하는 일이 벌어집니다.

- 결합 컬럼 인덱스를 index range scan 할 수 있는 방법은 무엇인가?
  ---> deptno를 선두 컬럼으로 두는게 아니라 sal을 선두 컬럼으로 두고 인덱스를 다시 생성하면 됩니다.
  ---> 드롭하고 인덱스 다시 생성하기

```
drop index emp_deptno_sal;

create index emp_sal_deptno
      on emp(sal, deptno);
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP            |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN           | EMP_SAL_DEPTNO |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

```
select sal, deptno, rowid
      from emp
      where sal >=0;

select sal, deptno, rowid
      from emp
      where sal >=0;
```

해보면

```
SAL   DEPTNO    ROWID
800     20      AAASCJAAHAAAAH+AAK
950     30      AAASCJAAHAAAAH+AAH
1100    20      AAASCJAAHAAAAH+AAM
1250    30      AAASCJAAHAAAAH+AAE
1250    30      AAASCJAAHAAAAH+AAI
1300    10      AAASCJAAHAAAAH+AAN
1500    30      AAASCJAAHAAAAH+AAG
1600    30      AAASCJAAHAAAAH+AAF
2450    10      AAASCJAAHAAAAH+AAC
2850    30      AAASCJAAHAAAAH+AAB
2975    20      AAASCJAAHAAAAH+AAD
3000    20      AAASCJAAHAAAAH+AAJ
3000    20      AAASCJAAHAAAAH+AAL
5000    10      AAASCJAAHAAAAH+AAA
```

인덱스 만들 때 SAL을 앞으로 뒀기 때문에 SAL이 ascending하게 정렬 된 것을 볼 수 있다.
그래서 자주 쓰는 컬럼에 따라 index를 만들때 자주쓰는 컬럼을 앞에 두는 것이 좋다.

```
select /*+ gather_plan_statistics index(emp emp_deptno_sal) */ ename, deptno, sal
            from emp
            where sal = 2975;
      SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

select sal, deptno, rowid
      from emp
      where deptno >=0;
```

설명 : index skip scan은 emp_deptno_sal 인덱스를 처음부터 끝까지 읽는 것은 index full scan과 같지만 중간중간 skip 할 수 있습니다. 10번부터 조회해서 월급이 2975가 있는지 찾아보고 20번도 조회해서 월급이 2975가 있는지 찾아보고 있으면 거기까지만 검색하고 나머지는 skip 합니다. 그리고 30번도 2975가 있는지 조회합니다.

```
select /*+ gather_plan_statistics index_ss(emp emp_deptno_sal) */ ename, deptno, sal
            from emp
            where sal = 2975;
      SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP            |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX SKIP SCAN            | EMP_DEPTNO_SAL |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------
```

- index skip scan이 효과를 보려면 결합 컬럼 인덱스의 첫번째 컬럼의 데이터의 종류가 몇가지 안되어야 효과를 봅니다.
- 부서 번호가 10, 20, 30 만 있을 때와
  부서 번호가 10, 20, 30, 40, 50 .... 1000까지 있을때
  전자가 skip 할 수 있는 확률이 더 높으므로 선두 컬럼의 데이터의 종류가 적어야 유리합니다.

■ index_asc와 index_desc 힌트를 사용해서 튜닝하는 방법 (문제 31 참조)
"order by 절을 너무 자주 사용하면 성능이 떨어집니다."
대용량 데이터 베이스 환경에서는 더욱 그렇다.

■ index merge scan
"두개의 인덱스를 동시에 사용하여 하나의 인덱스만 사용했을 때보다 더 좋은 성능을 보이는 스캔 방법"

Ex) @demo <--- 테스트 잘 되도록 초기화 하는 것

```
create index emp_deptno on emp(deptno);
create index emp_job on emp(job);
```

아래의 SQL의 실행계획을 확인하세요!

```
select /*+ gather_plan_statistics */ ename, deptno, job
      from emp
      where job = 'SALESMAN' and deptno = 30;
```

스스로 학습하는 옵티마이져여서 자리마다 실행계획이 다르다.
위와 같이 index가 두개일 때 '인덱스를 모두 사용해서 더 큰 시너지 효과를 일으켜보자' 라는게 index merge scan입니다.

```
select /*+ gather_plan_statistics and_equal(emp emp_job emp_deptno) */ ename, deptno, job
      from emp
      where job = 'SALESMAN' and deptno = 30;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------  실행순서
|   0 | SELECT STATEMENT             |            |      1 |        |      4 |00:00:00.02 |       5 |          5
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP        |      1 |      4 |      4 |00:00:00.02 |       5 |    4
|   2 |   AND-EQUAL                  |            |      1 |        |      4 |00:00:00.02 |       4 |               3
|*  3 |    INDEX RANGE SCAN          | EMP_JOB    |      1 |      4 |      4 |00:00:00.02 |       3 |         1
|*  4 |    INDEX RANGE SCAN          | EMP_DEPTNO |      1 |      6 |      4 |00:00:00.01 |       1 |    2
-------------------------------------------------------------------------------------------------------
```

버퍼의 개수는 5개입니다.

설명 : 위의 스캔 방법은 옛날 방법입니다. 이 스캔 방법보다 더 좋은 스캔방법이 나왔는데 그게 바로 index bitmap merge scan입니다.

■ 7. index bitmap merge scan

두개의 인덱스를 같이 스캔해서 시너지 효과를 보는 방법인것은 index merge scan과 동일하지만 bitmap을 이용해서 인덱스의 사이즈를 아주 많이 줄인다는 차이점이 있습니다.
그래서 인덱스를 스캔하는 시간이 짧아집니다.

Ex) 책의 내용을 검색하기 위해서 두개의 목차를 먼저 읽고 요약해서 목차를 A4 한페이지로 만들어 버리면 목차를 빠르게 검색할 수 있기 때문에 책(테이블)의 데이터를 찾기가 쉬워진다. (빨라집니다)

관련 힌트 : index_combine

```
select /*+ gather_plan_statistics index_combine(emp) */ ename, deptno, job
      from emp
      where job = 'SALESMAN' and deptno = 30;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |      1 |        |      4 |00:00:00.02 |       3 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP        |      1 |      4 |      4 |00:00:00.02 |       3 |
|   2 |   BITMAP CONVERSION TO ROWIDS       |            |      1 |        |      4 |00:00:00.02 |       2 |
|   3 |    BITMAP AND                       |            |      1 |        |      1 |00:00:00.01 |       2 |
|   4 |     BITMAP CONVERSION FROM ROWIDS   |            |      1 |        |      1 |00:00:00.01 |       1 |
|*  5 |      INDEX RANGE SCAN               | EMP_DEPTNO |      1 |        |      6 |00:00:00.01 |       1 |
|   6 |     BITMAP CONVERSION FROM ROWIDS   |            |      1 |        |      1 |00:00:00.01 |       1 |
|*  7 |      INDEX RANGE SCAN               | EMP_JOB    |      1 |        |      4 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------
```

버퍼의 개수가 3개로 merge scan보다 좋아졌다.

■ 8. index join 스캔 방법

인덱스들끼리 조인을 해서 바로 결과를 보고 테이블 엑세스는 따로 하지 않는 스캔 방식

힌트 : /*+ index_join(emp emp_deptno emp_job) */

@demo

```
create index emp_deptno on emp(deptno);
create index emp_job on emp(job);

select /*+ gather_plan_statistics index_join(emp emp_deptno emp_job) */
      deptno, job
      from emp
      where job = 'SALESMAN' and deptno = 30;
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name              | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                   |      1 |        |      4 |00:00:00.01 |       2 |       |       |          |
|*  1 |  VIEW              | index$_join$_001  |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|*  2 |   HASH JOIN        |                   |      1 |        |      4 |00:00:00.01 |       2 |  1610K|  1610K| 1095K (0)|
|*  3 |    INDEX RANGE SCAN| EMP_DEPTNO        |      1 |      4 |      6 |00:00:00.01 |       1 |       |       |          |
|*  4 |    INDEX RANGE SCAN| EMP_JOB           |      1 |      4 |      4 |00:00:00.01 |       1 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

- 통계를 코드로 구현

평균:

```
select avg(sal) from emp;
```

분산: 데이터의 퍼짐 정도

```
select variance(sal) from emp;
```

표준편차 : 분산에 루트씌운 값

```
select stddev(sal) from emp;
select sqrt(variance(sal)) from emp;
```

루트 : select sqrt(4) from dual;

11/19 (조인튜닝)

2020년 11월 19일 목요일 오전 9:57

■ SQL튜닝 기술

1. 인덱스 SQL튜닝 : 인덱스 엑세스 방법 8가지
2. 조인 SQL튜닝
3. 서브쿼리 SQL튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL튜닝

■ 2. 조인SQL튜닝

1. nested loop조인
2. hash 조인
3. sort merge 조인
4. 조인 순서의 중요성
5. outer조인 튜닝
6. 스칼라 서브쿼리를 이용한 조인
7. 조인을 내포한 DML문 튜닝
8. 고급 조인 문장 튜닝

■ 1. nested loop조인

SQL시간에 배웠던 조인 문법 : 1. 1999 ANSI 조인 문법

2. 오라클 조인 문법
   - equi join
   - non equi join
   - outer join
   - self join

SQL 튜닝시간에 배우는 조인 방법 3가지 (방법)?

1. nested look join 실행계획
2. hash join
3. sort merge join

조인 방법을 왜 알아야 하는가? 조인을 빠르게 하려면 알아야 합니다.

- 조인 튜닝시 중요한 튜닝방법 2가지? **1. 조인 순서를 조정해라**

First. ordered: from 절에서 기술한 테이블 순서대로 조인해라

Second. leading : leading 힌트안에 쓴 테이블 순서대로 조인해라

2. 조인 방법을 결정해라~

1. nested loop join : 적은양의 데이터를 조인할 때 유리한 조인 방법

   힌트 : use_nl

2. hash join : 대용량 데이터를 조인할 때 유리한 조인방법
   힌트 : use_hash
3. sort merge join : 대용량 데이터를 조인할 때 유리한 조인방법 hash join으로 수행되지 못하는 SQL에 사용하여 튜닝
   힌트 : use_merge

- 작은 테이블을 조인할 때는 쓸데없이 해쉬조인을 하면서 메모리를 과다하게 사용할 필요는 없습니다. 작은 테이블 조인할 때는 nested loop조인이 되게 하세요.

■ 해쉬조인(hash join)

nested loop조인은 순차적 반복 조인으로 한 레코드씩 순차적으로 조인을 진행하는 특징이 있습니다.

MARTIN CHICAGO
ALLEN CHICAGO
TURNER CHICAGO
WARD CHICAGO

순차적으로 하나씩 조인을 시도합니다. 처음에는 MARTIN의 데이터로 조인하려고 하고 그다음은 ALLEN으로 조인하려는 데이터의 양이 많으면 성능이 너무 느립니다.
그러다보니 대량의 데이터를 조인할 때는 순차적 반복을 여러 번 반복해야 하므로 비효율적입니다. 그래서 대량의 데이터를 조인할 때는 nested loop 조인보다는 해쉬(hash)조인을 사용하는 것이 훨씬 성능이 좋습니다.

해쉬조인은 해쉬 알고리즘을 이용해서 데이터를 메모리에 올려놓고 메모리에서 조인하는 조인 방법입니다. 메모리에서 다 조인하면 좋은데 메모리는 디스크에 비해서 크기가 아주 작습니다.
메모리는 크기가 아주 작기때문에 서로 메모리에 데이터 올려놓고 조인하겠다고 하면 경합이 걸려서 다 같이 느려집니다.
대용량 데이터를 조인할 때는 nested loop조인을 하면 성능이 너무 느리므로 해쉬조인을 수행하면 됩니다.

관련 힌트: use_hash

```
Ex ) select /*+ gather_plan_statistics leading (d e) use_hash(e) */
            e.ename, d.loc
            from emp e, dept d
            where e.deptno = d.deptno
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      16 |       |       |          |  (4)
|*  1 |  HASH JOIN         |      |      1 |     14 |     14 |00:00:00.01 |      16 |  1797K|  1797K| 1109K (0)|  (3)
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |  (1)
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       8 |       |       |          |  (2)
-------------------------------------------------------------------------------------------------------
```

- **실행순서**

dept 테이블과 emp 테이블이 둘다 메모리로 올라가는데 메모리에 둘다 한번에 다 올라가서 조인을 하면 너무 좋을텐데 그렇게는 안되고 둘중에 하나만 먼저 메모리로 올리고 그리고 다른 테이블은 데이터의 일부를 조금씩 메모리로 올리면서 조인을 합니다.

```
select rowid, deptno, loc
    from dept;

AAASAKAAHAAAAP+AAA    10      NEW YORK
AAASAKAAHAAAAP+AAB    20      DALLAS
AAASAKAAHAAAAP+AAC    30      CHICAGO
AAASAKAAHAAAAP+AAD    40      BOSTON
```

Δ

어느 파일에 어느 블럭이 있다라는 물리적 주소 정보

dept테이블이 디스크에 있을 때는 이 주소가 유용했는데 dept 테이블이 통째로 메모리에 올라가면(이사가면) 위의 주소가 필요 없어지고 새로운 주소가 필요한데 그 주소가 메모리의 주소입니다.

HASH JOIN

TABLE ACCESS FULL DEPT ---> hash table (메모리로 올라간 테이블)

TABLE ACCESS FUL EMP ---> probe table (디스크에 있는 테이블)

메모리가 공간이 작으므로 emp와 dept테이블 중 어느 테이블을 메모리로 올려야 할까요?

작은 테이블인 dept를 올려야 됩니다.

해쉬조인 주의사항 ? 1. 작은테이블이 메모리로 올라가게 힌트를 주면 됩니다.

Ex)

```
 select /*+ gather_plan_statistics leading (d e) use_hash(e) */
            e.ename, d.loc
            from emp e, dept d
            where e.deptno = d.deptno
```

해쉬조인도 작은 것을 먼저 올려야 합니다.

■ sort merge join

조인하려는 데이터의 양이 많으면서 조인조건이 =이 아닐때 검색 성능을 높이기 위한 조인 방법. sort merge조인을 수행하면 연결고리가 되는 키 컬럼(deptno)을 오라클이 알아서 정렬해 놓고 조인을 합니다.

```
Ex ) select /*+ gather_plan_statistcs leading(d e) use_merge(e) */
            e.ename, d.loc, e.deptno
       from emp e, dept d
       where e.deptno = d.deptno;
```

ENAME	LOC	DEPTNO
KING	NEW YORK	10
CLARK	NEW YORK	10
MILLER	NEW YORK	10
ADAMS	DALLAS	20
SCOTT	DALLAS	20
SMITH	DALLAS	20
FORD	DALLAS	20
JONES	DALLAS	20
WARD	CHICAGO	30
JAMES	CHICAGO	30
ALLEN	CHICAGO	30

MARTIN	CHICAGO	30
BLAKE	CHICAGO	30
TURNER	CHICAGO	30

결과를 보면 order by 절을 사용하지 않았는데 deptno가 정렬이 되어있습니다. 그 얘기는 sort merge join이 deptno를 정렬해놓고 조인을 수행했다는 것입니다. 정렬을 해서 데이터를 모아놨으므로 조인이 빨라집니다.

where d.deptno = e.deptno

10	10
20	10
30	10
40	10
	20
	20
	20
	.
	.

11/20

2020년 11월 20일 금요일 오전 9:44

■ (복습) SQL튜닝 목차

(핵심 튜닝)

1. 인덱스 튜닝
2. 조인문장 튜닝
3. 서브쿼리문 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동SQL튜닝

- 조인문장 튜닝 시 중요한 기술 2가지?
  1. 조인 순서:
     1) leading
     2) ordered
  2. 조인 방법:
     1) nested loop조인 : use_nl : 적은양의 데이터로 조인
     2) hash 조인 : use_hash : 대용량 데이터로 조인
     3) sort merge 조인: use_merge: 대용량 데이터로 조인

OLTP서버	DW 서버
↓	↓
실시간 조회 처리를 위한	Ex)건강보험 심사평가원
데이터가 저장되어 있는 곳	(우리나라 전 국민의 모든 의료기록 저장/
	10년전 정보, 20년전 정보도 모두 포함)
↓	↓
nested loop조인을 주로 사용	해쉬조인, merge조인 데이터 분석함수 주로 사용
	파티션 테이블, 병렬처리 주로 함

■ **4. 조인순서의 중요성**

- 조인순서에 따라서 수행 성능이 달라질 수 있습니다.
- 조인순서는 조인하려는 시도가 적은 테이블을 driving테이블로 선정해서 조인순서를 고려하는게 중요합니다.

nested loop

table access full dept ---> 선행 테이블, driving 테이블

table access full emp ---> 후행 테이블, driven 테이블

선행 테이블, driving 테이블

후행 테이블, driven테이블

Hash join

table access full dept --> hash table (메모리로 올라가는 테이블)

table access full emp --> probe table (디스크에 있는 테이블/ 탐색하는 테이블)

어떤 조인이든간에 둘다 먼저 읽는 테이블은 테이블의 크기가 작거나 검색조건절에 의해서 엑세스 되는 건수가 작은 것을 먼저 읽는게 조인순서에 있어서 중요합니다.

<조인문장 튜닝하는 순서>

2번

Ex) select /*+ gather_plan_statistics leading (d e) */ e.ename, d.loc
from emp e, dept d -- emp 14건, dept 4건 **1번**
where e.deptno = d.deptno;

i. 조인순서를 정한다. : from절의 테이블의 건수를 다 확인한다.

ii. 조인방법을 정한다. : 접속한 서버의 종류가 뭔지 확인한다.
(OLTP 서버인지 DW 서버인지 확인한다)
위와 같이 데이터 건수를 확인해서 대용량이 아니면 nested loop조인을 쓴다.
만약 중요한 SQL이고, OLTP라도 대용량이라 nested loop로 튜닝해도 효과가 없으면 hast join한다.

■ 5. outer조인의 튜닝 방법

- outer 조인의 조인순서는 outer조인 sign이 없는 쪽에서 있는 쪽으로 순서가 고정이 됩니다. 그러다보니 조인 순서를 변경하기가 어려워 튜닝이 힘든데 이를 개선할 수 있는 힌트가 있습니다.

Ex) insert into emp(empno, ename, sal, deptno)
values(2921, 'JACK', 4500, 70);

```
select /*+ gather_plan_statistics */ e.ename, d.loc
    from emp e, dept d
    where e.deptno = d.deptno (+);
            10          10
            20          20
            30          30
            70          40
```

설명 : 결과에 JACK이 출력되면서 부서위치가 비어있는게 확인이 됩니다.
JACK이 어느 부서위치에도 배치가 되지 않았구나라는 것을 알게 됩니다.
JACK이 결과로 출력되게 하려면 그냥 equi join으로 안되고 outer join을 사용해야 합니다.
실행계획을 확인해보면 join순서가 어떻게 되나요?

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |      14 |       |       |          |
|*  1 |  HASH JOIN OUTER   |      |      1 |     14 |     15 |00:00:00.01 |      14 |  1922K|  1922K|  986K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

설명 : 지금 보면 emp 테이블을 선행테이블로 읽었습니다. 왜냐하면 outer join은 outer 사인(sign)이 없는 쪽에서 있는 쪽으로 조인합니다. emp 테이블을 먼저 읽어야 emp에는 있지만 dept에는 없는 것을 알 수 있어서 emp테이블을 먼저 읽습니다.

■ 6. 스칼라 서브쿼리를 이용한 조인

- select문에서 서브쿼리를 쓸 수 있는 절

select ----> 서브쿼리 가능 : 스칼라 서브쿼리(scalar subquery)
from ----> 서브쿼리 가능 : in line view
where ----> 서브쿼리 가능
group by ----> X
having ----> 서브쿼리 가능
order by ----> 서브쿼리 가능 : 스칼라 서브쿼리(scalar subquery)

■ 7. 조인을 내포한 DML 문장 튜닝

- "DML문을 이용한 서브쿼리문의 튜닝"

Ex) ALLEN의 월급을 KING의 월급으로 변경하시오!

```
update emp
    set sal = (select sal
               from emp
               where ename = 'KING')
    where ename = 'ALLEN';
```

>>> 서브쿼리를 이용한 업데이트문

■ 8. 고급조인문장 튜닝

"뷰 안의 조인문장의 조인순서를 변경하고자 할 때 사용하는 튜닝방법"

Ex) @demo

```
drop view emp_dept;

create view emp_dept
as
    select e.empno, e.ename, e.sal, e.deptno, d.loc
        from emp e, dept d
        where e.deptno = d.deptno;
```

```
select *
    from emp_dept;
```

■ 서브쿼리문 튜닝

```
select ename, sal
    from emp
    where sal > (select sal
                 from emp
                 where ename = 'JONES');
```

서브쿼리 문장 튜닝의 기술은 크게 2가지를 알면 됩니다.

1. 순수하게 서브쿼리문으로 수행되게 하는 방법 : no_unnest
2. 서브쿼리를 조인으로 변경해서 수행되게 하는 방법 : unnest

nest : 감싸다

unnest : 감싸지 않고 풀어헤친다

no_unnest : 감싸라!!!!!! (강조)

11/23

2020년 11월 23일 월요일 오전 9:38

■ SQL튜닝 목차

1. 인덱스 SQL튜닝 (8가지 엑세스 방법)
    a. index range scan
    b. index unique scan
    c. index full scan
    d. index fast full scan
    e. index skip scan
    f. index merge scam
    g. index bitmap merge scan
    h. index join

2. 조인 문장 튜닝 (조인순서, 조인방법 3가지)
    a. nested loop 조인 : use_nl
    b. hash 조인 : use_hash
    c. sort merge 조인 : use_merge
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL 튜닝

■ 3. 서브쿼리 문장 튜닝

Ex) DALLAS에서 근무하는 사원들의 이름과 월급을 출력하시오!

```
select ename, sal
        from emp
        where deptno in ( select deptno
                                from dept
                                where loc = 'DALLAS' );
```

- 아래의 서브쿼리문의 실행계획을 보시오!

```
select /*+ gather_plan_statistics */ ename, sal
        from emp
        where deptno in (select deptno
                                from dept
                                where loc = 'DALLAS');
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      5 |00:00:00.01 |      12 |       |       |          |
|*  1 |  HASH JOIN SEMI    |      |      1 |      5 |      5 |00:00:00.01 |      12 |  1651K|  1651K| 1069K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|*  3 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       6 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

설명 : 위의 실행계획에 filter가 보이면 위의 SQL을 서브쿼리문으로 수행한 것이고 그게 아니고 hash join이 실행계획에 보이면 조인문장으로 변경해서 수행한 것입니다. 우리는 서브쿼리문으로 작성했는데 옵티마이져가 조인으로 변경해서 수행한 것입니다.

- 서브쿼리의 실행계획의 2가지 힌트

1. 순수하게 서브쿼리문으로 수행되게 하는 방법 : no_unnest (강하게 감싸라)
   1) 서브쿼리부터 수행해라~ : push_subq
   2) 메인쿼리부터 수행해라~ : no_push_subq

2. 조인으로 변경되어서 수행되게 하는 방법 : unnest (감싸지 말고 풀어헤치고 조인으로 변경해라)
   1) 세미조인 (semi join)
      One. nested loop semin join : nl_sj
      Two. hash semi join : hash_sj
      Three. merge semi join : merge_sj
   2) 안티조인 (anti join)
      One. nested loop anti join : nl_aj
      Two. hash anti join : hash_aj
      Three. merge anti join : merge_aj

- **서브쿼리 튜닝을 정리하면?**

  서브쿼리와 메인쿼리의 테이블이 대용량이 아니면?

  " 순수하게 서브쿼리 실행계획으로 실행되게 하세요~"

  관련힌트 ? no_unnest

  no_unnest와 짝꿍인 힌트? push_subq
  no_push_subq

  서브쿼리와 메인쿼리의 테이블이 대용량이면?

  " 해쉬세미 조인 또는 해쉬 안티 조인으로 수행되게 하세요~"

  (해쉬세미 조인)

```
select ename
    from emp
    where empno in (select mgr
                    from emp);
```

  관련 힌트 : /*+ unnest hash_sj */

  (해쉬안티 조인)

```
select ename
    from emp
    where empno not in (select mgr
                        from emp);
```

  관련 힌트 : /*+ unnest hash_aj */

■ 4. 데이터 분석함수를 이용한 튜닝

" 앞에서는 SQL 튜닝을 할 때 힌트를 이용해서 튜닝을 했는데 지금부터는 SQL을 완전히 다른 SQL로 변경

해서 튜닝하겠습니다."

■ 5. 자동 SQL 튜닝

SQL 튜닝 어드바이저(advisor) <<< 모를 때 마다 물어볼 수 있다. 기계라서..
완전하진 않지만 점점 인공지능화 되어가고 있다.

SQL 튜닝 어드바이저(advisor)에게 악성 SQL 튜닝을 물어보는 방법:

1. 테스트 테이블 test를 생성하고 10000개의 데이터를 입력합니다.

```
SQL> create table test (n number );
```

test 테이블에 숫자를 1번부터 1000번까지 입력하는 문장

```
declare
begin
for  i  in  1 .. 10000  loop
insert into test values(i);
commit;
end loop;
end;
/
```

```
select count(*)
    from test;

  COUNT(*)
----------
     10000
```

-- 인덱스 생성

```
create index test_idx on test(n);
```

-- test 테이블 통계정보 분석

(test 테이블에 대한 정보를 분석해서 오라클에게 알려주는 명령어)

>>>

1. **테이블의 건수**
2. **테이블의 크기**
3. **인덱스는 어디에 있는지... 등등의 정보**

```
analyze table test estimate statistics;
```

테이블이 분석되었습니다. 라고 나옴

-- NO_INDEX 힌트를 주어 풀테이블 스캔으로 수행되는 SQL확인

- **test 테이블의 n컬럼의 데이터가 1인 데이터를 검색하는데 no_index 힌트를 이용해서 test테이블의 test_idx 인덱스를 타지말고 full table scan해라 하고 실행합니다. 악성 SQL 만드려고 이부**

러 NO_INDEX힌트를 줬습니다.

```
select /*+ gather_plan_statistics NO_INDEX(test test_idx) */ *
           from test
           where n = 1 ;
```

이렇게하면

```
-------------------------------------------------------------------------------------
| Id  | Operation         | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |      1 |        |      9 |00:00:00.01 |       6 |
|*  1 |  TABLE ACCESS FULL| EMP12 |      1 |      2 |      9 |00:00:00.01 |       6 |
-------------------------------------------------------------------------------------
```

full scan 한다.

튜닝 TASK생성 후, SQL Tuning Advisor를 실행

- SQL 튜닝 어드바이져에게 악성 SQL을 주면서 튜닝해 달라고 부탁하는 명령어

```
declare
my_task_name VARCHAR2(30);
my_sqltext CLOB;
begin
my_sqltext := 'select /*+ no_index(test test_idx) */ *
from test where n = 1';   ---> 튜닝해야 할 SQL
my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK (
sql_text => my_sqltext,
user_name => 'SCOTT',   ----> 유져이름
scope => 'COMPREHENSIVE',
time_limit => 60,   -----> 60분 안에 일 다 마쳐라~
task_name => 'my_sql_tuning_task_1',  ----> 튜닝 작업이름
description => 'Task to tune a query on a specified table' );
end;
/
```

이 부분만 변경하면 됨

- 위에서 만든 튜닝 작업을 실행해라~

```
begin
DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
task_name => 'my_sql_tuning_task_1' );
end;
 /
```

- 튜닝분석결과를 확인하는 방법

```
SQL> SET LONG 70000
SQL> SET LONGCHUNKSIZE 1000
SQL> SET LINESIZE 100

SQL> select DBMS_SQLTUNE.REPORT_TUNING_TASK( 'my_sql_tuning_task_1') from DUAL;
```

레포트 결과를 볼 수 있습니다.

```
SQL> DECLARE
my_sqlprofile_name VARCHAR2(30);
BEGIN
my_sqlprofile_name := DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
task_name => 'my_sql_tuning_task_1',
name => 'my_sql_profile' );
END;
/
```

결과 :

```
DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------------
GENERAL INFORMATION SECTION
-------------------------------------------------------------------------------
Tuning Task Name   : my_sql_tuning_task_1
Tuning Task Owner  : SCOTT
Workload Type      : Single SQL Statement
Execution Count    : 2
Current Execution  : EXEC_231
Execution Type     : TUNE SQL
Scope              : COMPREHENSIVE
Time Limit(seconds): 60
Completion Status  : COMPLETED

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
------------------------------------------------------------------------------------------------------- 이만큼 걸렸다
Started at         : 11/23/2020 16:38:57
Completed at       : 11/23/2020 16:38:57

-------------------------------------------------------------------------------
Schema Name: SCOTT
SQL ID     : a6p7tf37qf0fu
SQL Text   : select /*+ no_index(test test_idx) */ *
             from test where n = 1

------------------------------------------------------------------------------- 위의 SQL을 수행했다.
FINDINGS SECTION (1 finding)

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------

1- SQL Profile Finding (see explain plans section below)
--------------------------------------------------------
  이 명령문에 대해 잠재적으로 더 나은 실행 계획이 발견되었습니다.

  Recommendation (estimated benefit: 90.92%)
  ------------------------------------------
  - 권장 SQL 프로파일 수용을 고려하십시오.
    execute dbms_sqltune.accept_sql_profile(task_name =>
            'my_sql_tuning_task_1', task_owner => 'SCOTT', replace => TRUE);

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------------

  Validation results
  ------------------
  SQL profile이(가) 해당 계획과 원래 계획을 모두 수행하고 각각의 수행 통계를 평가하여 테스트되었습니
다. 더 짧은 시간 내에
```

완료될 수 있는 실행이 있었다면 계획이 부분적으로만 수행되었을 수 있습니다.

```
                         Original Plan  With SQL Profile  % Improved
                         -------------  ----------------  ----------
  Completion Status:            COMPLETE          COMPLETE
  Elapsed Time (s):             .000144           .000022       84.72 %

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------
  CPU Time (s):                 .000634           .000022       96.52 %
  User I/O Time (s):                  0                 0
  Buffer Gets:                       22                 2        90.9 %
  Physical Read Requests:             0                 0
  Physical Write Requests:            0                 0
  Physical Read Bytes:                0                 0
  Physical Write Bytes:               0                 0
  Rows Processed:                     1                 1
  Fetches:                            1                 1
  Executions:                         1                 1


DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
------------------------------------------------------------------------------------------------- 이만큼 효율을 볼 수 있다.
  Notes
  -----
  1. the original plan에 대한 통계는 10회 실행에 대한 평균으로 계산되었습니다.
  2. the SQL profile plan에 대한 통계는 10회 실행에 대한 평균으로 계산되었습니다.

--------------------------------------------------------------------------------
EXPLAIN PLANS SECTION
--------------------------------------------------------------------------------

1- Original With Adjusted Cost
------------------------------

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------
Plan hash value: 1357081020

-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     3 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| TEST |     1 |     3 |     7   (0)| 00:00:01 |
-----------------------------------------------------------------------

Predicate Information (identified by operation id):
--------------------------------------------------- 오리지널 플랜

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------

   1 - filter("N"=1)

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 3 (U - Unused (3))
---------------------------------------------------------------------------

   0 -  STATEMENT
          U -  IGNORE_OPTIM_EMBEDDED_HINTS / hint overridden by another in parent query block
          U -  OPTIMIZER_FEATURES_ENABLE(default) / hint overridden by another in parent query block


DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------
   1 -  SEL$1 / TEST@SEL$1
          U -  no_index(test test_idx) / rejected by IGNORE_OPTIM_EMBEDDED_HINTS

2- Using SQL Profile
--------------------
Plan hash value: 2882402178

-----------------------------------------------------------------------
```

```
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |          |     1 |     3 |     1   (0)| 00:00:01 |

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------
|*  1 |  INDEX RANGE SCAN| TEST_IDX |     1 |     3 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------  개선된 플랜

   1 - access("N"=1)

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))
---------------------------------------------------------------------------

DBMS_SQLTUNE.REPORT_TUNING_TASK('MY_SQL_TUNING_TASK_1')
-------------------------------------------------------------------------------------------------

   1 -  SEL$1 / TEST@SEL$1
         U -   no_index(test test_idx) / rejected by IGNORE_OPTIM_EMBEDDED_HINTS

---------------------------------------------------------------------------
```

위의 악성 SQL을 그냥 둔 상태에서 그냥 실행계획인 인덱스 스캔하는 실행계획으로 바꿔치기 하고 싶다면?

튜닝레포트에 SQL profile 적용하는 문장을 수행하면 됩니다.

```
DECLARE
my_sqlprofile_name VARCHAR2(30);
BEGIN
my_sqlprofile_name := DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
task_name => 'my_sql_tuning_task_1',
name => 'my_sql_profile' );
END;
/
```

그리고 나서 아래의 SQL의 실행계획을 보세요~

```
 select /*+ gather_plan_statistics NO_INDEX(test test_idx) */ *
        from test
        where n = 1 ;
```

오늘의 문제 (001-126)

2020년 11월 16일 월요일 오후 4:27

1. **아래의 SQL을 튜닝하시오!**

   select empno, ename, sal, job
   from emp
   where empno = 7788;

   튜닝 전 : 실행계획 + 버퍼의 개수
   튜닝 후 :실행계획 + 버퍼의 개수

   튜닝 전:

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

   튜닝 후:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP       |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN           | EMP_EMPNO |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------
```

   (번외문제) SQL 알고리즘 22번 : 자연상수 e를 SQL로 구현하시오~

   select

2. **아래의 SQL을 튜닝하시오. 튜닝전과 튜닝후의 실행계획을 비교해보세요.**
   (특히 buffer의 개수를 확인하세요)
   buffer의 갯수는 그 SQL을 처리하기 위해서 읽어들인 메모리 출력의 개수 입니다.
   create index emp_job on emp(job);
   튜닝 전 : select /*+ gather_plan_statistics */ename, sal, job
   from emp
   where substr(job, 1, 5) = 'SALES';

   SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      4 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      4 |      4 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

   튜닝 후:

```
 select /*+ gather_plan_statistics */ ename, sal, job
                    from emp
                    where job like 'SALES%';
            SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |      1 |        |      4 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |      1 |      4 |      4 |00:00:00.01 |       2 |
```

```
|*  2 |   INDEX RANGE SCAN                | EMP_JOB |      1 |      4 |      4 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------
```

설명 : 좌변을 가공하면 인덱스 스캔을 할 수 없습니다.

튜닝전의

where substr(job, 1, 5) = 'SALES'; 에서

좌변 우변

따라서 좌변을 튜닝해주면 된다.

3. 아래의 SQL을 튜닝하시오!

create index emp_sal on emp(sal);

튜닝 전 : select /*+ gather_plan_statistics */ ename, sal

from emp

where sal*12 = 36000;

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      2 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      2 |      2 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

튜닝 후:

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where sal=36000/12;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |      1 |        |      2 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |      1 |      2 |      2 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_SAL |      1 |      2 |      2 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

4. 아래의 SQL을 튜닝하시오!

"좌변을 가공하면 안된다!"

create index emp_hiredate on emp(hiredate);

튜닝전 : select /*+ gather_plan_statistics */ ename, hiredate

from emp

where to_char(hiredate, 'RRRR') = '1980';

SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      1 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

튜닝후:

```
select /*+ gather_plan_statistics index(emp emp_hiredate) */ ename, hiredate
                  from emp
where hiredate between to_date('1980/01/01', 'RRRR/MM/DD')
                  and to_date('1980/12/31', 'RRRR/MM/DD');
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |      1 |        |      1 |00:00:00.01 |       2 |
|*  1 |  FILTER                             |              |      1 |        |      1 |00:00:00.01 |       2 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| EMP          |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  3 |    INDEX RANGE SCAN                 | EMP_HIREDATE |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

5. 아래의 SQL을 튜닝하시오!

" 좌변을 가공하면 안된다!!!"

create index emp_empno on emp(empno);

튜닝 전 : select empno, ename, sal

from emp

where empno||ename = '7788SCOTT';

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      1 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |
```

```
-------------------------------------------------------------------------------
```

튜닝 후:

```
select /*+ gather_plan_statistics * index(emp emp_empno) */ empno, ename, sal
    from emp
    where empno = 7788 and ename = 'SCOTT';
   SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |           |      1 |        |      1 |00:00:00.01 |       2 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP       |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_EMPNO |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------
```

6. 아래의 SQL을 튜닝하시오!

튜닝 전:

```
select /*+ gather_plan_statistics */ ename, sal
       from emp
       where sal = '3000';
```

숫자형 = 문자형

Δ

숫자형

3000

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

❖ 설명 : 오라클이 암시적으로 문자형을 숫자형으로 변경해주었다. 숫자형이 더 우선순위가 높기 때문입니다.

튜닝 후:

```
select /*+ gather_plan_statistics */ ename, sal
       from emp
       where sal = 3000;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

7. 아래의 SQL을 튜닝하시오

튜닝 전 : select /*+ gather_plan_statistics */ ename, sal

from emp

where sal like '30%';

숫자형 문자형

SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      2 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      2 |      2 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

```
explain plan for
select ename, sal
    from emp
    where sal like '30%';
select * from table(dbms_xplan.display);
```

해보면

1 - filter(TO_CHAR("SAL") LIKE '30%')

이러한 문구를 볼 수 있다.

❖ 설명 : 위의 SQL의 경우도 오라클이 암시적으로 형변환을 해주었습니다. SQL에서 조건절에 LIKE를 자주 사용하는 쿼리의 테이블 컬럼은 테이블 생성시부터 숫자형이 아니라 문자형으로 만들어 줬어야 합니다.

튜닝 후: 함수기반 인덱스를 생성해서 튜닝합니다.

```
create index emp_sal_func
       on emp(to_char(sal));
select */+ gather_plan_statistics * index(emp emp_empno) */ ename, sal
    from emp
    where sal like '30%';
select * from table(dbms_xplan.display);
```

```
select /*+ gather_plan_statistics * index(emp emp_empno) */ ename, sal
      from emp
      where sal between 3000 and 3999;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |      1 |        |      2 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |      1 |      2 |      2 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_SAL |      1 |      2 |      2 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

8. (점심시간 문제) (SQL 알고리즘 문제 23번) 구글입사문제
1부터 10000까지의 숫자중에 숫자 8이 총 몇번 나오는가?
8이 포함되어 있는 숫자의 개수를 카운팅하는게 아니라 8이라는 숫자를 모두 카운팅 해야합니다.
(예를 들어 8808은 3, 8888은 4로 카운팅 해야 함)
힌트 : 계층형 질의문, regexp_count를 이용하면 쉽게 할 수 있습니다. 또는 decode와 sum등을 이용합니다.

결과: 4000
정답 :

```
select regexp_count(level,8) "1부터 10000까지의 8의 개수"
      from dual
      connect by level <=10000;
```

9. 다음 결과에서 최댓값을 출력하시오.
보기:

```
select decode(job, 'PRESIDENT', null, sal)
      from emp;
```

답:

```
select max(decode(job, 'PRESIDENT', null, sal))
      from emp;
```

- 설명 : 3000을 예상했는데 950이 나온 이유는 암시적 형변환이 발생했기 때문입니다. decode(job, 'PRESIDENT', null, sal)을 사용하게 되면 decode의 세번째 인자값이 null 이면 네번째 인자값의 출력이 문자형으로 암시적 형변환이 발생합니다.
- Ex)
```
select to_char(sal) as salary
      from emp
      order by salary desc;
```
- 설명 : 이렇게 하면 950이 젤 위로 올라갑니다. 월급을 문자형으로 변환하게 되면 숫자 9가 앞에 있는 값이 제일 큰 값이 됩니다. 문자형으로 봤을 때는1이 젤 작고 9가 젤 크기 때문입니다.

10. 아래의 SQL을 수정해서 원래 예상했던 값이 출력되게 암시적 형변환이 일어나지 않도록 SQL을 수정하시오

보기:

```
select max(decode(job, 'PRESIDENT', null, sal))
      from emp;
```

답:

```
select max(decode(job, 'PRESIDENT', 0, sal))
      from emp;
```

11. 사원 이름에 non unique 인덱스를 걸고 이름이 SCOTT인 사원의 이름과 월급을 조회하는 쿼리문의 실제 실행계획을 확인하시오!

```
select /*+ gather_plan_statistics */ ename, sal
   from emp
   where ename = 'SCOTT';
     SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |           |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP       |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_ENAME |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

설명 : 인덱스를 읽을 때 인덱스를 2개 이상 읽으면 index range scan이 됩니다. 하나만 읽으면 index unique scan입니다.
그런데 왜 SCOTT은 데이터가 하나인데 왜 SCOTT하나만 읽지 못하고 SMITH까지 읽었을까?
왜냐하면 emp_ename인덱스가 non unique인덱스로 생성되었기 때문입니다.

12. 사원 테이블에 직업에 인덱스를 걸고 아래의 SQL을 수행해서 index range scan으로 실행계획이 나오는지 확인하시오!

```
create index emp_job on emp(job);

select /*+ gather_plan_statistics */ ename, job
       from emp
       where job = 'SALESMAN';
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |      1 |        |      4 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | EMP_JOB |      1 |      4 |      4 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

13. 사원번호가 7788번인 사원의 사원번호와 사원이름을 출력하는 쿼리문의 실행계획을 확인하시오!

```
select /*+ gather_plan_statistics index(emp emp_empno) */ empno, ename
       from emp
       where empno = 7788;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

❖ 설명: unique인덱스는 인덱스에서 딱 한개의 데이터만 읽고 끝낸다. 그러므로 non unique 인덱스보다 훨씬 좋은 인덱스 입니다.

14. **dept테이블의 deptno에 primary key제약을 걸고 dept 테이블의 unique인덱스가 자동으로 생성되었는지 확인하시오!**

```
alter table dept
       add constraint dept_deptno_pk primary key(deptno);
```

15. **dept테이블의 loc에 non unique인덱스를 생성하시오.**

```
create index dept_loc
    on dept(loc);
```

16. **아래의 SQL을 튜닝하시오.**

(인덱스를 엑세스하고 테이블을 엑세스 하게 하시오)

튜닝 전 : select /*+gather_plan_statistics */ *
from dept
where deptno = 20 and loc = 'DALLAS';
unique index non unique index 로 설정된 상태

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |      1 |        |      1 |00:00:00.01 |       2 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| DEPT           |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX UNIQUE SCAN         | DEPT_DEPTNO_PK |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

unique 인덱스를 탔다. 그래서 따로 튜닝 안해줘도 됨.

17. **아래의 SQL이 loc에 걸린 non unique인덱스를 엑세스 하도록 힌트를 주시오!**

```
select /* gather_plan_statistics index(dept dept_loc) */ *
       from dept
       where deptno = 20 and loc='DALLAS';
    SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |      1 |        |      1 |00:00:00.01 |       2 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| DEPT     |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX RANGE SCAN                  | DEPT_LOC |      1 |      2 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

18. **위의 SQL이 full table scan이 되도록 힌트를 주시오!**

```
select /*+ gather_plan_statistics full(emp) */ count(*)
       from emp;
```

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |       7 |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------------
```

이렇게 나온다.

설명 : full table scan은 table갯수가 3이고 index full scan은 버퍼의 개수가 1로 3배의 성능이 좋아지는 효과를 볼 수 있습니다.

19. **우리반 테이블의 ename에 인덱스를 걸고 우리반 학생들의 인원수를 출력하는 쿼리문의 성능을 높일 수 있게끔 SQL을 구현하시오.**

❖ 설명 :

index_fs(emp emp_empno)힌트가 index full scan이고 full(emp)*/ count(*)이게 index full scan입니다.

(??? 필기 못함)

```
create index emp12_ename on emp12(ename);

select /*+ gather_plan_statistics index_fs(emp12 emp12-ename) */ count(ename)
       from emp12;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-------------------------------------------------------------------------------------
| Id  | Operation          | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |              |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT AGGREGATE    |              |      1 |      1 |      1 |00:00:00.01 |       1 |
|   2 |   INDEX FULL SCAN| EMP12_ENAME |      1 |     30 |     30 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------
```

20. **부서번호, 부서번호별 인원수를 출력하는데 index fast full scan이 될 수 있도록 인덱스도 걸고 not null 제약도 걸고 힌트도 줘서 실행되게 하시오.**

```
create index emp_deptno
   on emp(deptno);

alter table emp
   modify de서ptno constraint emp_deptno_nn not null;

select /*+ gather_plan_statistics index_ffs(emp emp_deptno) */ deptno, count(deptno)
   from emp
   group by deptno;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

21. **우리반 테이블에서 통신사, 통신사별 인원수를 출력하는데 index fast full scan이 될 수 있도록 하시오!**

```
create index emp12_telecom
on emp12(telecom);

alter table emp12
   modify telecom constraint emp12_telecom_nn not null;

select /*+ gather_plan_statistics index_ffs(emp12 emp12_telecom) */ telecom, count(telecom)
   from emp12
   group by telecom;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

22. **(오늘 마지막 문제) (SQL알고리즘 24번) 구글문제와 비슷**

2^100 (2^{100})의 각 자리 수의 합은?

2^15 = 32768 의 자릿수를 다 더하면

3+2+7+6+8 = 26 입니다.

```
select sum(substr(power(2,100), level, 1)) "2^100의 자릿수의 합"
   from dual
   connect by level <= 100;
```

근데 여기에 length쓰는 것이 더 좋음

23. **아래의 쿼리가 인덱스를 통해서만 데이터를 가져오고 테이블 엑세스를 하지 않도록 결합컬럼 인덱스를 생성하시오.**

```
select /*+ gather_plan_statistics */ empno, ename, sal, deptno
      from emp
      where empno = 7788;
```

정답 :

```
create index empno_ename_sal_deptno
   on emp(empno, ename, sal, deptno);

select /*+ gather_plan_statistics index(emp empno_ename_sal_deptno) */ empno, ename, sal, deptno
            from emp
where empno = 7788;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation         | Name                  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                       |      1 |        |      1 |00:00:00.01 |       1 |
|*  1 |  INDEX RANGE SCAN| EMPNO_ENAME_SAL_DEPTNO |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------
```

Index 이름이 너무 길면 인덱스가 만들어지지 않을 수 있어서 index1 이런식으로 이름 지어주는 것이 좋다.

24. **@demo 스크립트를 다시 돌리고 아래의 SQL을 튜닝하시오!**

index fast full scan이 되도록 튜닝하시오!

(1. 인덱스 생성, 2. not null 제약 건다. 3. 힌트주고 쿼리 작성)

```
select /*+ gather_plan_statistics */ job, count(job)
      from emp
      group by job;
```

1. 인덱스 생성

```
create index emp_job
      on emp(job);
```

2. not null 제약 생성

```
alter table emp
   modify job constraint emp_job_nn not null;
```

3. 쿼리문 작성

```
select /*+ gather_plan_statistics index_ffs(emp emp_job) */ job, count(job)
   from emp
   group by job;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

25. **다시 demo 스크립트를 돌리고 아래의 SQL이 index fast full scan이 되도록 튜닝하시오!**
(1. 인덱스 생성, 2. not null 제약 생성, 3. 힌트를 사용한 쿼리문 작성)

튜닝전 :

```
select /*+ gather_plan_statistics */ deptno, count(empno)
      from emp
      group by deptno;
```

1. 인덱스 생성

```
create index empno_deptno
   on emp(empno, deptno);
```

2. not null제약 형성

```
alter table emp
   modify deptno constraint emp_deptno_nn not null;
```

(not null은 둘 중 하나에만 걸어줘도 됨)

설명 : not null 제약을 걸어주는 이유는 인덱스를 목차로 보면 목차 페이지에 빈페이지가 없다는 것을 보장하기 위해서 입니다.

3. 힌트를 사용한 쿼리문 작성

```
select /*+ gather_plan_statistics index_ffs(emp empno_deptno) */ deptno, count(empno)
      from emp
```

```
        group by deptno;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               |      1 |        |      3 |00:00:00.02 |       4 |       |       |          |
|   1 |  HASH GROUP BY        |               |      1 |     14 |      3 |00:00:00.02 |       4 |  1558K|  1558K|  772K (0)|
|   2 |   INDEX FAST FULL SCAN| EMPNO_DEPTNO  |      1 |     14 |     14 |00:00:00.01 |       4 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------
```

26. **@demo를 돌리고 아래의 인덱스를 생성하고 아래의 SQL을 튜닝하시오!**
(min과 group by를 사용하지 않고 똑같은 결과가 출력되게 하시오.)

create index emp_deptno_sal on emp(deptno, sal);

힌트 : 결합 컬럼 인덱스의 구조를 이해하면 풀 수 있는 문제입니다.

튜닝전 : select /*+ gather_plan_statistics */ deptno, min(sal)
from emp
where deptno = 20
group by deptno;

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT GROUP BY NOSORT |                |      1 |      1 |      1 |00:00:00.01 |       1 |
|*  2 |   INDEX RANGE SCAN    | EMP_DEPTNO_SAL |      1 |      1 |      5 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------
```

설명 : 실행계획에 sort라는 말이 나오면서 데이터를 정렬했다고 나오는 실행계획은 좋은 실행계획은 아닙니다. 정렬하는데 시간이 걸리기 때문입니다.

```
select /*+ gather_plan_statistics index_asc(emp emp_deptno_sal)*/ deptno, sal
                    from emp
                    where deptno = 20 and rownum = 1;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-------------------------------------------------------------------------------------------
| Id  | Operation         | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |      1 |        |      1 |00:00:00.01 |       1 |
|*  1 |  COUNT STOPKEY    |                |      1 |        |      1 |00:00:00.01 |       1 |
|*  2 |   INDEX RANGE SCAN| EMP_DEPTNO_SAL |      1 |      5 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------
```

Count Stopkey : 여기까지 읽고 멈췄다.

27. **아래의 SQL을 튜닝하시오!**

튜닝 전 : select /* gather_plan_statistics */ deptno, max(sal)
from emp
where deptno = 20
group by deptno;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

튜닝 후:

```
select /*+ gather_plan_statistics index_desc(emp emp_deptno_sal)*/ deptno, sal
                    from emp
                    where deptno = 20 and rownum = 1;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

28. **아래의 SQL을 튜닝하시오!**

튜닝전 : select job, max(hiredate)
from emp
where job = 'SALESMAN'
group by job;

튜닝 후 :

```
create index emp_job_hiredate
    on emp(job, hiredate);

select /*+ gather_plan_statistics index_desc(emp emp_job_hiredate) */ job, hiredate
    from emp
    where job = 'SALESMAN' and rownum = 1;
```

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                   |      1 |        |      1 |00:00:00.01 |       1 |
|*  1 |  COUNT STOPKEY               |                   |      1 |        |      1 |00:00:00.01 |       1 |
|*  2 |   INDEX RANGE SCAN DESCENDING| EMP_JOB_HIREDATE  |      1 |      4 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

- **(점심시간 문제)**

통계를 코드로 구현 : (이항분포1). 한개의 주사위를 360번 던져서 3의 배수의 눈이 나올 확률을 구하시오!

```
select count(*)/360
    from (select round(dbms_random.value(0.5, 6.5)) 주사위
            from dual
            connect by level <=360)
    where mod(주사위, 3) = 0;
```

29. **다시 @demo 스크립트를 수행하고 아래의 SQL이 index skip scan이 되도록 튜닝하시오!**

@demo <--- 기존 테이블이 drop되고 다시 생성되면서 기존 인덱스들이 다 사라집니다.

create index emp_job_sal on emp(job, sal);

(튜닝 전)

```
select /*+ gather_plan_statistics */ empno, ename, job, sal
    from emp
    where sal = 1250;
```

(튜닝 후)

```
select /*+ gather_plan statistics index_ss(emp emp_job_sal) */ empni, ename, job, sal
    from emp
    where sal = 1250;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |      1 |        |      2 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP         |      1 |      2 |      2 |00:00:00.01 |       2 |
|*  2 |   INDEX SKIP SCAN                   | EMP_JOB_SAL |      1 |      1 |      2 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

30. **직업의 종류가 몇가지 입니까?**

결과 : 숫자 5

```
select count(distinct job)
   from emp;
```

31. **@demo를 돌리고 아래의 인덱스를 생성한 후 아래의 SQL을 튜닝하시오!**

```
@demo
create index emp_sal on emp(sal);
```

튜닝전 :
```
select /*+ gather_plan_statistics */ ename, sal
        from emp
        order by sal desc;
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |       7 |       |       |          |
|   1 |  SORT ORDER BY     |      |      1 |     14 |     14 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------
```

튜닝후 :

```
select /*+ gather_plan_statistics index_desc(emp emp_sal)*/ ename, sal
from emp
where sal >= 0;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |        |      1 |        |     14 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP    |      1 |     14 |     14 |00:00:00.01 |       2 |
```

```
|*  2 |   INDEX RANGE SCAN DESCENDING       | EMP_SAL |      1 |      14 |      14 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------
```

설명 : index_desc 힌트가 작동하려면 인덱스 컬럼이 where절에 검색조건으로 있어줘야 합니다.

32. **아래의 SQL을 튜닝하시오! (실행계획에 SORT가 있지 않게 하시오)**
max 함수 사용하지 말고 인덱스에서 데이터를 읽어오게 튜닝하시오.
튜닝전 : select /*+ gather_plan_statistics */ max(sal)
from emp;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT AGGREGATE             |         |      1 |      1 |      1 |00:00:00.01 |       1 |
|   2 |   INDEX FULL SCAN (MIN/MAX)| EMP_SAL |      1 |      1 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

튜닝 후 :

```
select /*+ gather_plan_statistics index_desc(emp emp_sal) */ sal
      from emp
      where sal >=0 and rownum = 1;
      SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |      1 |        |      1 |00:00:00.01 |       1 |
|*  1 |  COUNT STOPKEY               |         |      1 |        |      1 |00:00:00.01 |       1 |
|*  2 |   INDEX RANGE SCAN DESCENDING| EMP_SAL |      1 |     14 |      1 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------
```

33. **아래의 SQL을 튜닝하시오! (emp 테이블을 한번만 엑세스해서 결과가 나오게 하시오)**
튜닝 전:

```
select ename, sal
      from emp
      where sal = (select max(sal)
                   from emp) ;
```

튜닝후 :

```
select /*+ gather_plan_statistics index_desc(emp emp_sal) */  ename, sal
      from emp
      where sal >=0 and rownum = 1;
```

이 조건을 넣어줘야 인덱스를 탄다.

버퍼의 개수가 줄었다.

34. **@demo를 돌리고 아래의 SQL을 튜닝하시오! (order by절을 쓰지 않고 출력되게 하시오!)**

```
@demo
create index emp_deptno_sal on emp(deptno, sal);
```

튜닝전 :

```
select /*+ gather_plan_statistics */ empno, deptno, sal
            from emp
            where deptno = 20
            order by sal desc;
```

튜닝후 :

```
 select /*+ gather_plan_statistics index_desc(emp emp_deptno_sal) */ empno, deptno, sal
                  from emp
                  where deptno = 20;
```

35. **@demo를 돌리고 아래의 SQL을 튜닝하시오!**
(인덱스도 직접 생성하세요)
튜닝 전 : select /*+ gather_plan_statistics */ ename, hiredate
from emp
where to_char(hiredate, 'RRRR') = 1981 <<< 좌변에 함수 있으면 안됨
order by hiredate desc;

튜닝후:

1. 인덱스 생성

```
create index emp_hiredate
```

on emp(hiredate);

2. 쿼리 생성

```
select /*+ gather_plan_statistics index_desc(emp emp_hiredate) */ ename, hiredate
    from emp
    where hiredate between to_date('1981/01/01', 'RRRR/MM/DD')
          and to_date('1981/12/31', 'RRRR/MM/DD');
```

36. **@demo를 돌리고 아래의 SQL을 튜닝하시오!**
(결합 컬럼 인덱스를 직접 생성하세요!)

@demo
튜닝전 : select /*+ gather_plan_statistics */ ename, job, sal
from emp
where substr(job, 1, 5) = 'SALES'
order by sal desc;

튜닝후 :

1. 인덱스 생성

```
create index emp_job_sal
    on emp(job, sal);
```

2. 쿼리 생성

```
select /*+ gather_plan_statistics index_desc(emp emp_job_sal) */ ename, job, sal
   from emp
   where job like 'SALES%';
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

37. **다음의 SQL이 deptno 걸린 인덱스를 타도록 힌트를 주고 실행하시오!**

@demo

```
create index emp_deptno on emp(deptno);
create index emp_job on emp(job);
```

아래의 SQL의 실행계획을 확인하세요!

```
select /*+ gather_plan_statistics */ ename, deptno, job
    from emp
    where job = 'SALESMAN' and deptno = 30;
```

정답:

```
select /*+ gather_plan_statistics index(emp emp_deptno) */ ename, deptno, job
    from emp
    where job = 'SALESMAN' and deptno = 30;
```

38. **37번 문제 연관 - 이번에는 job에 걸린 인덱스를 타도록 힌트를 주고 실행하시오!**

```
select /*+ gather_plan_statistics index_fs(emp emp_job) */ ename, deptno, job
    from emp
    where job = 'SALESMAN' and deptno = 30;
```

설명 : job과 deptno 단일 컬럼 인덱스를 각각 걸었기에 위의 SQL의 조건절에서는 두개의 컬럼을 다 사용해서 데이터를 검색하고 있습니다.
그러면 2개 중 어느 하나의 인덱스를 사용해야 하는데 어떤 컬럼의 인덱스를 사용하는게 더 성능에 좋을까요?
---> 목차를 길게 읽는것보다 목차를 짧게 읽는게 더 좋은 목차(인덱스)

각각의 개수를 파악해보자.

```
select count(*)
    from emp
    where job = 'SALESMAN';
    ---> 4개
select count(*)
    from emp
    where deptno = 30;
    ---> 6개
```

- **(오늘의 마지막 문제) 통계를 코드로 구현**

평균:

select avg(sal) from emp;

분산: 데이터의 퍼짐 정도

select variance(sal) from emp;

표준편차 : 분산에 루트씌운 값

select stddev(sal) from emp;

select sqrt(variance(sal)) from emp;

루트 : select sqrt(4) from dual;

39. **이항 분포 공식을 이용해서 다음 이항 분포의 평균값과 분산값과 표준편차를 구하시오.**
한개의 주사위를 36000번 던져서 3의 배수의 눈이 나올 확률의 횟수를 확률변수 X 라고 하자. 이때 확률변수 X의 평균값과 분산값과 표준편차를 구하시오.

이항분포의 평균값 : np : 36000 x 1/3 = 12000

이항분포의 분산값 : npq : 36000 x 1/3 x 2/3 = 8000

이항분포의 표준편차 : $\sqrt{npq} = \sqrt{8000}$

$= \sqrt[40]{5}$

위의 이항분포의 평균값과 분산값과 표준편차를 수학공식을 이용하지 말고 컴퓨터를 이용해서 SQL코드로 구하시오!

위의 공식으로 구한 결과의 근사값으로 출력됩니다.

40. **사원 이름, 월급, 부서위치를 출력하는데 조인 순서가 어떤게 더 성능이 좋겠는가?**
    1. **emp---------->dept**
    2. **dept---------->emp**

설명 : 2번 순서가 더 성능이 좋다. 작은 테이블부터 먼저 읽고 큰 테이블이랑 조인하는게 더 성능이 좋다.

```
select e.ename, e.sal, d.loc
   from emp e, dept d
   where e.deptno = d.deptno;
```

41. **위의 조인문장의 실행계획을 확인해서 어느 테이블을 먼저 읽고 조인했는지 확인하시오!**

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
   from emp e, dept d
   where e.deptno = d.deptno;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |          |      1 |        |     14 |00:00:00.01 |       8 |       |       |          |
|  1 |  MERGE JOIN                  |          |      1 |     14 |     14 |00:00:00.01 |       8 |       |       |          |
|  2 |   TABLE ACCESS BY INDEX ROWID| EMP      |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|  3 |    INDEX FULL SCAN           | EMP_DEPTNO |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|* 4 |   SORT JOIN                  |          |     14 |      4 |     14 |00:00:00.01 |       6 |  2048 |  2048 | 2048  (0)|
|  5 |    TABLE ACCESS FULL         | DEPT     |      1 |      4 |      4 |00:00:00.01 |       6 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
```

설명 : 실행계획을 봤을때 dept 테이블 부터 먼저 읽고 emp와 조인을 했으면 잘한겁니다.

버퍼의 개수는 14개이고, emp테이블을 먼저 읽었습니다.

42. 문제 41번의 조인문장의 조인순서를 emp----> dept순으로 조인되게 순서를 조정하시오!

- **조인 순서를 조정하는 힌트? ordered 힌트**

ordered힌트는 from 절에서 기술한 테이블 순서대로 조인해라

```
select /*+ gather_plan_statistics ordered */ e.ename, e.sal, d.loc
   from emp e, dept d
   where e.deptno = d.deptno;
```

설명 : 조인 순서를 emp -----> dept순으로 변경했더니 버퍼의 개수 62개로 증가했습니다.

43. **다시 위의 SQL의 조인 순서를 dept ---> emp순이 되도록 힌트를 주고 SQL을 작성하시오.**

```
 select /*+ gather_plan_statistics ordered */ e.ename, e.sal, d.loc
      from dept d, emp e
      where e.deptno = d.deptno;
 SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      15 |       |       |          |
|* 1 |  HASH JOIN         |      |      1 |     14 |     14 |00:00:00.01 |      15 |  1797K|  1797K|  969K (0)|
|  2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|  3 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

이렇게 하거나

다른 튜닝방법 : 기존 튜닝전 SQL에 힌트가 있으면 힌트를 먼저 빼고 수행해본다.

44. (조인순서 연습) emp와 salgrade 테이블을 조인해서 이름과 월급과 급여등급(grade)을 출력하시오!
(조인 순서를 성능이 좋도록 하시오)

```
select /*+ gather_plan_statistics ordered*/ e.ename, e.sal, s.grade
    from salgrade s, emp e
    where e.sal between s.losal and s.hisal;
 SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
--------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     14 |00:00:00.01 |      13 |       |       |          |
|   1 |  MERGE JOIN          |          |      1 |      1 |     14 |00:00:00.01 |      13 |       |       |          |
|   2 |   SORT JOIN          |          |      1 |      5 |      5 |00:00:00.01 |       6 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL | SALGRADE |      1 |      5 |      5 |00:00:00.01 |       6 |       |       |          |
|*  4 |   FILTER             |          |      5 |        |     14 |00:00:00.01 |       7 |       |       |          |
|*  5 |    SORT JOIN         |          |      5 |     14 |     40 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
--------------------------------------------------------------------------------------------------------------------
```

❖ 조인 순서를 결정하는 힌트 2가지?

1. ordered : from절에서 기술한 테이블 순서대로 조인해라~
2. leading : leading 힌트 안에 사용한 테이블 순서대로 조인해라

```
select /*+ gather_plan_statistics leading(e s)*/ e.ename, e.sal, s.grade
    from salgrade s, emp e
    where e.sal between s.losal and s.hisal;
```

```
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
--------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     14 |00:00:00.01 |      13 |       |       |          |
|   1 |  MERGE JOIN          |          |      1 |      1 |     14 |00:00:00.01 |      13 |       |       |          |
|   2 |   SORT JOIN          |          |      1 |     14 |     14 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL | EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
|*  4 |   FILTER             |          |     14 |        |     14 |00:00:00.01 |       6 |       |       |          |
|*  5 |    SORT JOIN         |          |     14 |      5 |     40 |00:00:00.01 |       6 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       6 |       |       |          |
--------------------------------------------------------------------------------------------------------------------
```

```
 select /*+ gather_plan_statistics leading(s e)*/ e.ename, e.sal, s.grade
        from salgrade s, emp e
        where e.sal between s.losal and s.hisal;
```

45. **emp와 dept 테이블을 조인해서 이름과 월급과 직업과 부서위치를 출력하시오!**
leading 힌트를 이용해서 조인순서를 정하시오!

emp ----------> dept

```
select /*+ gather_plan_statistics leading(e d)*/ e.ename, e.sal, e.job, d.loc
    from emp e, dept d
    where e.deptno = d.deptno;
     SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

46. 문제 45번의 쿼리의 실행계획이 nested loop 조인이 되게 힌트를 주시오!

```
select /*+ gather_plan_statistics leading(e d) use_nl(e d) */ e.ename, e.sal, e.job, d.loc
    from emp e, dept d
    where e.deptno = d.deptno;
     SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |     105 |
|   1 |  NESTED LOOPS      |      |      1 |     14 |     14 |00:00:00.01 |     105 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |
|*  3 |   TABLE ACCESS FULL| DEPT |     14 |      1 |     14 |00:00:00.01 |      98 |
-----------------------------------------------------------------------------------------
```

버퍼의 개수: 105개

설명 : 해쉬조인 했을때와는 다르게 버퍼가 47개로 더 늘어났습니다. 그렇지만 emp와 dept테이블이 서로 조인되는 데이터의 양이 작으므로 nested loop조인을 사용하는게 더 바람직합니다.

47. **문제46번 쿼리의 조인 문장의 실행계획이 nested loop조인이 되도록 하되 조인순서는 dept ---->emp 순이 되도록 조인하시오!**

```
select /*+ gather_plan_statistics leading(d e) use_nl(d e) */ e.ename, e.sal, e.job, d.loc
    from emp e, dept d
    where e.deptno = d.deptno;
     SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      35 |
|   1 |  NESTED LOOPS      |      |      1 |     14 |     14 |00:00:00.01 |      35 |
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |
```

```
|*  3 |   TABLE ACCESS FULL| EMP  |      4 |      4 |     14 |00:00:00.01 |      28 |
-------------------------------------------------------------------------------------------
```

leading과 use_nl은 함께 바꾸어 주어야한다./ 조인 순서와 방법을 바꾸는 두가지를 같이 써주는게 좋다.
버퍼 : 35개

설명 : 17개로 버퍼가 줄어들었습니다.

48. emp와 salgrade테이블을 조인해서 이름과 월급과 급여등급(grade)을 출력하는 조인문장의 조인 순서와 조인 방법을 아래와 같이 되게 하시오!

salgrade ----------> emp
Δ
nested loop조인

```
select /*+ gather_plan_statistics leading(s e) use_nl(s e) */ e.ename, e.sal, s.grade
          from emp e, salgrade s
          where e.sal between s.losal and s.hisal;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |      1 |        |     14 |00:00:00.01 |      41 |
|   1 |  NESTED LOOPS      |          |      1 |      1 |     14 |00:00:00.01 |      41 |
|   2 |   TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       6 |
|*  3 |   TABLE ACCESS FULL| EMP      |      5 |      1 |     14 |00:00:00.01 |      35 |
-------------------------------------------------------------------------------------------
```

49. 이름이 SCOTT인 사원의 이름과 월급과 부서위치를 출력하는 조인문장을 작성하시오!

```
select e.ename, e.sal, d.loc
     from emp e, dept d
     where e.deptno = d.deptno and e.ename = 'SCOTT';
```

50. 위의 SQL은 조인순서가 아래의 2개중에 어떤게 더 좋은 순서인가?

1. **emp ----------> dept**
2. **dept ----------> emp**

`

정답 : 1번

설명 : emp 테이블에서 먼저 이름이 SCOTT인 사원의 데이터를 1건만 읽고 dept테이블과 1번만 조인하면 된다. SCOTT의 부서위치만 알면 되기 때문에 조인 시도 횟수가 1번입니다.

아래와 같은 상황일 때

```
select e.ename, e.sal, d.loc
     from emp e, dept d
     where e.deptno = d.deptno;
```

1. emp ----------> dept : 14번 조인시도
2. dept ----------> emp : 4번 조인시도

51. 아래의 조인문장의 조인순서는 emp-------->dept로 조인하고 조인방법은 nested loop 조인이 되도록 힌트를 주고 튜닝하시오!

```
select /*+ gather_plan_statistics leading(e d), use_nl(e d) */ e.ename, e.sal, d.loc
     from emp e, dept d
     where e.deptno = d.deptno and e.ename = 'SCOTT';
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |      16 |       |       |          |
|*  1 |  HASH JOIN         |      |      1 |      1 |      1 |00:00:00.01 |      16 |  1651K|  1651K|  402K (0)|
|*  2 |   TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       8 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

설명 : 조인문장에 조인(연결)조건 말고 검색조건이 따로 있으면 검색조건으로 검색되는 건수가 몇건인지 먼저 확인을 하고 그 건수가 조인 대상이 되는 상대쪽 테이블보다 작다면 작은 건수를 검색하는 쪽을 먼저 읽어야 합니다.

52. 아래의 SQL의 조인방법은 무조건 nested loop 조인으로 하되 조인순서는 자신이 직접 정하시오!

```
select e.ename, e.sal, d.loc
     from emp e, dept d
     where e.deptno = d.deptno
```

```
        and e.job = 'SALESMAN'
        and d.loc= 'CHICAGO';
```

튜닝 후:

```
select /*+ gather_plan_statistics leading(d e) use_nl(d e) */ e.ename, e.sal, d.loc
          from emp e, dept d
          where e.deptno = d.deptno
          and e.job = 'SALESMAN'--4건 이렇게 쓰는건 주석 달기라서 이렇게 해도 실행에 문제X
           and d.loc= 'CHICAGO'; --1건
```

53. (점심시간 문제) 아래의 SQL이 적절한 실행계획이 나올 수 있도록 튜닝하시오!

(조인방법은 무조건 nested loop로 하세요~)

튜닝 전:

```
select e.ename, e.sal, s.grade
      from emp e, salgrade s
      where e.sal between s.losal and s.hisal
      and s.grade =1
```

튜닝 후:

```
select /*+ gather_plan_statistics leading(s e) use_nl(e) */ e.ename, e.sal, s.grade
   from emp e, salgrade s
   where e.sal between s.losal and s.hisal
      and s.grade =1;
```

설명 : leading 힌트에 사용했던 테이블 별칭 중 두번째 별칭을 use_nl에 기술하면 됩니다.
leading(s e) use_nl(e)의 뜻은 salgrade와 emp 순으로 조인하는데 emp와 조인할 때 nested loop 조인해라~ 라는 뜻입니다.

54. emp와 dept와 salgrade를 조인해서 이름과 월급과 부서위치와 급여등급을 출력하시오! (SQL문제)

```
select e.ename, e.sal, d.loc, s.grade
      from emp e, dept d, salgrade s
      where e.deptno = d.deptno and e.sal between s.losal and s.hisal;
```

55. 위의 SQL의 조인 순서와 방법을 아래와 같이 되게 하시오!

dept -----> emp -----> salgrade << 연결고리가 서로 있는 테이블끼리 연결하면서 조인 순서를
Δ Δ
nested loop nested loop 정해줘야 합니다.

원래 크기로보면 dept ----> salgrade ----> emp 이렇게 순서를 정하고 싶지만 dept와
4개 5개 14개 salgrade가 서로 연결고리가 없어서
이렇게 하면 안된다.

```
select/*+ gather_plan_statistics leading(d e s) use_nl(e) use_nl(s) */ e.ename, e.sal, d.loc, s.grade
      from emp e, dept d, salgrade s
      where e.deptno = d.deptno and e.sal between s.losal and s.hisal;
```

설명 : dept ---> emp ---> salgrade순으로 조인하면서 dept와 emp와 조인할 때 nested loop join하고 dept와 emp와 조인한 결과와 salgrade와 조인할 때 nested loop조인해라 라는 뜻

56. 이번에는 아래와 같이 조인되게 하시오!

salgrade ---> emp ---> dept
Δ Δ
nested loop nested loop

```
select /*+ gather_plan_statistics leading(s e d)  use_nl(e)  use_nl(d) */
       e.ename, e.sal, d.loc, s.grade
  from  emp  e,  dept  d, salgrade   s
  where  e.deptno = d.deptno
    and  e.sal  between  s.losal   and  s.hisal;
```

57. 문제 56번의 조인문장의 실행계획을 분석하시오.

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------  실행순서
|   0 | SELECT STATEMENT   |          |      1 |        |     14 |00:00:00.01 |     139 |     6
|   1 |  NESTED LOOPS      |          |      1 |      1 |     14 |00:00:00.01 |     139 |     5
|   2 |   NESTED LOOPS     |          |      1 |      1 |     14 |00:00:00.01 |      41 |     3
|   3 |    TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       6 |  1
|*  4 |    TABLE ACCESS FULL| EMP      |      5 |      1 |     14 |00:00:00.01 |      35 |     2
|*  5 |   TABLE ACCESS FULL | DEPT     |     14 |      1 |     14 |00:00:00.01 |      98 |     4
-------------------------------------------------------------------------------------
```

58. 이름이 king인 사원의 이름과 월급과 부서위치와 급여등급을 출력하시오! (SQL문제)

```
select e.ename, e.sal, d.loc, s.grade
    from emp e, dept d, salgrade s
    where e.deptno = d.deptno <--- 조인(연결) 조건
        and e.sal between s.losal and s.hisal <--- 조인(연결) 조건
        and e.ename = 'KING'; <--- 검색 조건 1건
```

조인순서 : **emp** ---> dept ---> salgrade

젤 첫번째로 갖다 놓는 테이블이 검색속도를 좌우한다.

설명 : emp테이블에서 ename이 KING인 데이터가 1건이므로 1건만 읽어서 dept테이블과 조인하고 dept 테이블과 조인해 부서위치 가져오고 emp테이블과 조인한 결과가 salgrade테이블과 조인합니다.

59. 위의 SQL의 조인 순서를 아래와 같이 하고 조인방법은 전부 nested loop 조인하시오.

조인순서 : emp ---> dept ---> salgrade

```
select /*+ gather_plan_statistics leading (e d s) use_nl(d), use_nl(s) */ e.ename, e.sal, d.loc, s.grade
    from emp e, dept d, salgrade s
    where e.deptno = d.deptno
    and e.sal between s.losal and s.hisal
        and e.ename = 'KING';
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |        |      1 |00:00:00.01 |      20 |
|   1 |  NESTED LOOPS       |           |      1 |      1 |      1 |00:00:00.01 |      20 |      5
|   2 |   NESTED LOOPS      |           |      1 |      1 |      1 |00:00:00.01 |      14 |      3
|*  3 |    TABLE ACCESS FULL| EMP       |      1 |      1 |      1 |00:00:00.01 |       7 |    1
|*  4 |    TABLE ACCESS FULL| DEPT      |      1 |      1 |      1 |00:00:00.01 |       7 |    2
|*  5 |   TABLE ACCESS FULL | SALGRADE  |      1 |      1 |      1 |00:00:00.01 |       6 |  4
-----------------------------------------------------------------------------------------
```

60. CHICAGO에서 근무하는 사원들의 이름과 부서위치와 월급과 급여등급을 출력하시오!

(SQL문제)

```
select e.ename, d.loc, e.sal, s.grade
    from emp e, dept d, salgrade s
    where e.deptno = d.deptno and e.sal between s.losal and s.hisal
    and d.loc = 'CHICAGO';
```

61. 위의 SQL의 조인방법은 전부 nested loop조인으로 하되 조인순서를 직접 줘서 튜닝하시오!

(힌트를 주세요~)

답:

조인순서 : dept ---> emp ---> salgrade

(dept와 salgrade는 연결고리가 없으니...)

```
select /*+ gather_plan_statistics leading( d e s) use_nl(e) use_nl(s) */ e.ename, d.loc, e.sal, s.grade
    from emp e, dept d, salgrade s
    where e.deptno = d.deptno and e.sal between s.losal and s.hisal
    and d.loc = 'CHICAGO';
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |        |      6 |00:00:00.01 |      50 |
|   1 |  NESTED LOOPS       |           |      1 |      1 |      6 |00:00:00.01 |      50 |
|   2 |   NESTED LOOPS      |           |      1 |      5 |      6 |00:00:00.01 |      14 |
|*  3 |    TABLE ACCESS FULL| DEPT      |      1 |      1 |      1 |00:00:00.01 |       7 |
|*  4 |    TABLE ACCESS FULL| EMP       |      1 |      5 |      6 |00:00:00.01 |       7 |
|*  5 |   TABLE ACCESS FULL | SALGRADE  |      6 |      1 |      6 |00:00:00.01 |      36 |
-----------------------------------------------------------------------------------------
```

62. 부서위치, 부서위치별 토탈월급을 출력하시오!

```
select d.loc, sum(e.sal)
    from emp e, dept d
    where e.deptno = d.deptno
    group by d.loc;
```

63. 문제 62번의 SQL의 조인순서와 조인방법을 기술하고 수행하시오!

조인순서 : dept ---> emp

```
select /*+ gather_plan_statistics leading(d e), use_nl(e) */ d.loc, sum(e.sal)
    from emp e, dept d
    where e.deptno = d.deptno
    group by d.loc;
```

64. 아래의 SQL을 튜닝하는데 해쉬조인 되게하고 emp테이블이 메모리로 올라가서 hash테이블이 되게 하세요.

```
 select /*+ gather_plan_statistics leading(e d) use_hash(d) */
            e.ename, d.loc
            from emp e, dept d
            where e.deptno = d.deptno;
```

앞에 있는 emp table이 hash table 에 올라가게 됨

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      16 |       |       |          |
|*  1 |  HASH JOIN         |      |      1 |     14 |     14 |00:00:00.01 |      16 |  1856K|  1856K| 1033K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       8 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

65. 직업이 SALESMAN인 사원들의 이름과 월급과 comm2를 출력하시오!
(emp와 bonus테이블을 조인하세요)

```
select e.ename, e.sal, b.comm2
       from emp e, bonus b
       where e.empno = b.empno;
```

66. 위의 SQL의 실행계획을 아래와 같이 나오게 하시오!

조인순서 : emp ----------> bonus
                Δ
                해쉬조인

```
select /*+ gather_plan_statistics leading(e b) use_hash(b) */ e.ename, e.sal, b.comm2
       from emp e, bonus b
       where e.empno = b.empno;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |     14 |00:00:00.01 |      11 |       |       |          |
|*  1 |  HASH JOIN         |       |      1 |     14 |     14 |00:00:00.01 |      11 |  1610K|  1610K| 1280K (0)|
|   2 |   TABLE ACCESS FULL| EMP   |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| BONUS |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

버퍼의 개수 11개

설명 : 위와같이 emp와 bonus 둘다 14건으로 둘중에 작은 테이블이 없다면 조건에 의해서 걸러지는 데이터가 작은것을 메모리로 올리면 됩니다. 직업이 SALESMAN인 사원들의 데이터가 4건이므로 4건만 메모리로 올리고 bonus와 조인하면 됩니다.

67. 아래의 조인문장의 조인순서와 조인방법을 직접 결정하시오 (조인방법은 hash조인으로 하세요)

```
select e.ename, d.loc
       from emp e, dept d
       where e.deptno = d.deptno
       and e.job = 'SALESMAN'
       and d.loc = 'CHICAGO';
```

조인순서: dept ----> emp

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
       from emp e, dept d
       where e.deptno = d.deptno
       and e.job = 'SALESMAN' -- 4건
       and d.loc = 'CHICAGO' -- 1건;
```

68. emp와 dept와 bonus를 조인해서 이름과 부서위치와 comm2를 출력하시오! (SQL문제)

```
select e.ename, d.loc, b.comm2
       from emp e, dept d, bonus b
       where e.deptno = d.deptno and e.empno = b.empno;
```

69. 위의 SQL의 조인순서와 조인방법을 아래와 같이 하시오!

조인순서 : dept ---> emp ---> bonus
                해쉬조인   해쉬조인

```
select /*+ gather_plan_statistics leading (d e b) use_hash(e) use_hash(b) */
 e.ename, d.loc, b.comm2
       from emp e, dept d, bonus b
       where e.deptno = d.deptno and e.empno = b.empno;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |     14 |00:00:00.01 |      19 |       |       |          |     (6)
|*  1 |  HASH JOIN         |       |      1 |     14 |     14 |00:00:00.01 |      19 |  1476K|  1476K| 1282K (0)|     (5)
|*  2 |   HASH JOIN        |       |      1 |     14 |     14 |00:00:00.01 |      15 |  1797K|  1797K| 1047K (0)|     (3)
```

```
|  3 |   TABLE ACCESS FULL| DEPT  |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |           (1)
|  4 |   TABLE ACCESS FULL| EMP   |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |           (2)
|  5 |   TABLE ACCESS FULL | BONUS |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |           (4)
-----------------------------------------------------------------------------------------------------------------------------
```

설명 : dept와 emp와 hash join하고 그리고 두개를 해쉬조인한 결과와 보너스와 해쉬조인 했다.

HASH JOIN
HASH JOIN
 TABLE ACCESS FULL DEPT ---> hash table
 TABLE ACCESS FULL EMP ---> probe table
 TABLE ACCESS FULL BONUS --->probe table

70. 위 SQL의 실행계획을 아래와 같이 나오게 하시오.

(보너스가 Hash table로 올라가게)

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |      1 |        |     14 |00:00:00.01 |      19 |       |       |          |
|*  1 |  HASH JOIN          |       |      1 |     14 |     14 |00:00:00.01 |      19 |  1995K|  1995K| 1559K (0)|
|   2 |   TABLE ACCESS FULL | BONUS |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|*  3 |   HASH JOIN         |       |      1 |     14 |     14 |00:00:00.01 |      16 |  1797K|  1797K|  953K (0)|
|   4 |    TABLE ACCESS FULL| DEPT  |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|   5 |    TABLE ACCESS FULL| EMP   |      1 |     14 |     14 |00:00:00.01 |       8 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
```

```
select /*+ gather_plan_statistics leading (d e b)
                              use_hash(e) use_hash(b) swap_join_inputs(b) */
 e.ename, d.loc, b.comm2
      from emp e, dept d, bonus b
      where e.deptno = d.deptno and e.empno = b.empno;
```

힌트 : swap_join_inputs : hash table을 결정하는 힌트
no_swap_join_inputs : probe table을 결정하는 힌트

2개의 테이블을 조인할 때 hash table을 결정하는것은 leading 힌트만으로도 가능했는데 3개 이상의테이블을 조인 할 때 hash table을 결정 할 때는 leading으로는 제어가 안되고swap_join_inputs를 사용해야 합니다.
bonus테이블이 크기가 작다면 이 실행계획이 좋은 실행계획입니다.

71. **아래와 같이 실행계획이 나오게 하시오!**

```
select /*+ gather_plan_statistics leading (b e d)
                              use_hash(e) use_hash(d) swap_join_inputs(d) */
 e.ename, d.loc, b.comm2
      from emp e, dept d, bonus b
      where e.deptno = d.deptno and e.empno = b.empno;
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |      1 |        |     14 |00:00:00.01 |      19 |       |       |          |
|*  1 |  HASH JOIN          |       |      1 |     14 |     14 |00:00:00.01 |      19 |  1797K|  1797K| 1061K (0)|   (5)
|   2 |   TABLE ACCESS FULL | DEPT  |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |   (4)
|*  3 |   HASH JOIN         |       |      1 |     14 |     14 |00:00:00.01 |      11 |  1995K|  1995K| 1452K (0)|   (3)
|   4 |    TABLE ACCESS FULL| BONUS |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |   (1)
|   5 |    TABLE ACCESS FULL| EMP   |      1 |     14 |     14 |00:00:00.01 |       8 |       |       |          |   (2)
-----------------------------------------------------------------------------------------------------------------------------
```

설명 : bonus테이블과 emp와 해쉬조인을 먼저 하고 이 해쉬조인한 결과와 dept와 해쉬조인 하는데 dept를 메모리로 올려서 해쉬테이블로 구성하고 해쉬조인을 수행했다. 3개의 테이블을 조인하는것이고 dept테이블을 메모리로 올리기 위해서 swap_join_inputs라는힌트를 사용했습니다.
swap_join_inputs는 해쉬조인때만 사용할 수 있습니다.

72. emp와 salgrade 테이블을 조인해서 이름과 월급과 급여등급을 출력하고 hash조인으로 유도하시오!

```
select /*+ gather_plan_statistics leading (s e)
                              use_hash(e) */
 e.ename, e.sal, s.grade
      from emp e, salgrade s
      where e.sal between s.losal and s.hisal;
```

실행계획:

```
-----------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     14 |00:00:00.01 |      43 |
|   1 |  NESTED LOOPS       |          |      1 |      1 |     14 |00:00:00.01 |      43 |
|   2 |   TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       7 |
|*  3 |   TABLE ACCESS FULL| EMP      |      5 |      1 |     14 |00:00:00.01 |      36 |
-----------------------------------------------------------------------------------------
```

설명 : 위의 SQL은 실행계획이 hash join으로 풀리지 않습니다. 왜냐하면 해쉬조인이 되려면 where절의 조인조건이 이퀄(=)이어야 됩니다. non equi join은 해쉬조인으로 수행되지 않습니다.

그렇다면 위와 같이 SQL의 두 테이블이 둘다 대용량인데 해쉬조인을 못쓰면 어떻게 튜닝을 해야 하는가?

"sort merge join을 사용하면 됩니다."

73. 아래의 SQL의 실행계획이 SORT MERGE조인이 되게하시오.

```
select /*+ gather_plan_statistics leading(s e) use_merge(e) */
       e.ename, e.sal, s.grade
  from emp e, salgrade s
  where e.sal between s.losal and s.hisal;
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     14 |00:00:00.01 |      13 |       |       |          |
|   1 |  MERGE JOIN          |          |      1 |      1 |     14 |00:00:00.01 |      13 |       |       |          |
|   2 |   SORT JOIN          |          |      1 |      5 |      5 |00:00:00.01 |       6 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL | SALGRADE |      1 |      5 |      5 |00:00:00.01 |       6 |       |       |          |
|*  4 |   FILTER             |          |      5 |        |     14 |00:00:00.01 |       7 |       |       |          |
|*  5 |    SORT JOIN         |          |      5 |     14 |     40 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------------
```

번외 문제: 통계를 코드로 구현 - 이항분포


```
select 300*count(동전1*동전2)/300 as 평균,
       300*count(동전1*동전2)/300*(1-count(동전1*동전2)/300) as 분산,
       sqrt(300*count(동전1*동전2)/300*(1-count(동전1*동전2)/300)) as 표준편차
   from( select round(dbms_random.value(0,1)) 동전1, round(dbms_random.value(0,1)) 동전2
           from dual
           connect by level <= 300)
       where 동전1 = 0 and 동전2= 0;
```

74. 아래의 조인문장의 조인순서와 조인방법을 결정하시오!

```
select /*+ gather_plan_statistics leading(e d) use_nl(d) */ e.ename, d.loc, e.sal
       from emp e, dept d
       where e.deptno = d.deptno
       and e.ename = 'KING';
```

조인 순서 : emp ------> dept

75. emp 와 dept가 대용량 테이블이라고 가정하고 아래의 SQL의 조인순서와 조인방법을 결정하시오!
(emp는 12000만건, dept는 5400건)

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
       from emp e, dept d
       where e.deptno = d.deptno;
```

조인순서 : d----e

설명 : sort merge조인은 내부적으로 정렬을 일으키는 단점이 있어서 굳이 deptno를 정렬해서 볼 필요가 없다면 hash조인을 사용하세요. (정렬하는데 시간이 걸리기 때문)

76. 아래의 SQL의 조인순서와 조인방법을 결정하시오!
(emp와 salgrade가 대용량 테이블이라고 가정하고 DW서버에서 수행되는 SQL이라고 가정하세요!)

```
select e.ename, e.sal, s.grade
       from emp e, salgrade s
```

where e.sal between s.losal and s.hisal
order by e.sal asc;

답 :

```
select /*+ gather_plan_statistics leading(s e) use_merge(e) */ e.ename, e.sal, s.grade
      from emp e, salgrade s
      where e.sal between s.losal and s.hisal;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

조인순서 : s---->e

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     14 |00:00:00.01 |      14 |       |       |          |
|   1 |  MERGE JOIN          |          |      1 |      1 |     14 |00:00:00.01 |      14 |       |       |          |
|   2 |   SORT JOIN          |          |      1 |      5 |      5 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL | SALGRADE |      1 |      5 |      5 |00:00:00.01 |       7 |       |       |          |
|*  4 |   FILTER             |          |      5 |        |     14 |00:00:00.01 |       7 |       |       |          |
|*  5 |    SORT JOIN         |          |      5 |     14 |     40 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

77. **아래의 조인순서를 dept ---> emp순이 되게하시오~**

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
      from emp e, dept d
      where e.deptno = d.deptno(+)
```

이렇게 해도 조인순서는 바뀌지 않는다.

설명 : 조인순서를 변경하려고 leading힌트를줬지만 조인순서가 변경되지 않았습니다. 왜냐하면 outer 조인은 조인순서가 outer join사인 없는 쪽에서 있는 쪽으로 고정이 되어있기 때문입니다.

78. **위의 outer 조인의 조인순서를 dept ---> emp순이 되게 하시오~**

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) swap_join_inputs(d) */ e.ename, d.loc
      from emp e, dept d
      where e.deptno = d.deptno(+);
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |      1 |        |     15 |00:00:00.01 |      14 |       |       |          |
|*  1 |  HASH JOIN RIGHT OUTER|      |      1 |     14 |     15 |00:00:00.01 |      14 |  1797K|  1797K| 1060K (0)|
|   2 |   TABLE ACCESS FULL  | DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL  | EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

설명 : swap_join_inputs를 사용해서 dept테이블을 hash 테이블로 구성을해서 dept테이블을 먼저 드라이빙 할 수 있도록 했습니다. 실행계획을 보면 hash right outer 조인으로 수행되었습니다.
outer조인을 튜닝하려면 hash조인으로 수행해야 합니다. 왜냐하면 swap_join_input이 해쉬조인시에만 사용할 수 있기 때문입니다.

79. **emp와 bonus를 조인해서 이름과 comm2를 출력하는데 outer join을 이용해서 JACK도 출력될 수 있도록 하시오! (SQL문제)**

```
select e.emp, b.comm2
      from emp e, bonus b
      where e.empno = b.empno(+);
```

80. **위의 조인순서가 bonus ---> emp순이 되게 하시오!**

```
select /*+ gather_plan_statistics leading (b e) use_hash(e) swap_join_inputs(b) */ e.emp, b.comm2
      from emp e, bonus b
      where e.empno = b.empno(+);
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

81. **이름, 월급, 사원 테이블에서의 최대월급을 출력하시오!**

(select절의 서브쿼리인 스칼라 서브쿼리를 이용해서)

튜닝 전:

```
select /*+ gather_plan_statistics */ ename, sal, (select max(sal)
                        from emp) 최대월급
      from emp;
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |       7 |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |
|   3 |  TABLE ACCESS FULL | EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

튜닝 후:

```
select /*+ gather_plan_statistics */
        ename, sal, max(sal) over () 최대월급
        from emp;
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |       7 |       |       |          |
|   1 |  WINDOW BUFFER     |      |      1 |     14 |     15 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------
```

설명 : emp 테이블을 튜닝전에는 두번 엑세스 했는데 튜닝 후에는 emp테이블을 한번만 엑세스해도 똑같은 결과를 출력할 수 있게 되었습니다.

82. **아래의 SQL을 튜닝하시오!**

튜닝 전: select /*+ gather_plan_statistics */ ename, sal, (select max(sal) from emp) 최대월급,
(select min(sal)from emp) 최소월급
from emp;

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |       7 |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     15 |     15 |00:00:00.01 |       7 |
|   3 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.01 |       7 |
|   4 |   TABLE ACCESS FULL| EMP  |      1 |     15 |     15 |00:00:00.01 |       7 |
|   5 |  TABLE ACCESS FULL | EMP  |      1 |     15 |     15 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

튜닝 후 : select /*+ gather_plan_statistics */
ename, sal, max(sal) over () 최대월급, min(sal) over () 최소월급
from emp;

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |       7 |       |       |          |
|   1 |  WINDOW BUFFER     |      |      1 |     14 |     15 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     15 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------
```

83. **아래의 SQL을 튜닝하시오!**
부서번호, 이름, 월급, 자기가 속한 부서번호의 평균 월급을 출력하는데 자기의 월급이 자기가 속한 부서번호의 평균월급보다 더 큰 사원들만 출력하시오!

튜닝 전 : select /*+ gather_plan_statistics */ e.deptno, e.ename, e.sal, v.부서평균
from emp e, (select deptno, avg(sal) 부서평균
from emp
group by deptno) v
where e.deptno = v.deptno and e.sal > v.부서평균;

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |      6 |00:00:00.01 |      10 |       |       |          |
|   1 |  MERGE JOIN                  |            |      1 |      3 |      6 |00:00:00.01 |      10 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP        |      1 |     15 |     15 |00:00:00.01 |       3 |       |       |          |
|   3 |    INDEX FULL SCAN           | EMP_DEPTNO |      1 |     15 |     15 |00:00:00.01 |       1 |       |       |          |
|*  4 |   FILTER                     |            |     15 |        |      6 |00:00:00.01 |       7 |       |       |          |
|*  5 |    SORT JOIN                 |            |     15 |     15 |     15 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   6 |     VIEW                     |            |      1 |     15 |      4 |00:00:00.01 |       7 |       |       |          |
|   7 |      SORT GROUP BY           |            |      1 |     15 |      4 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   8 |       TABLE ACCESS FULL      | EMP        |      1 |     15 |     15 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------
```

튜닝 후: emp 테이블을 한번만 엑세스해서 똑같은 결과가 출력되게 하시오

```
select *
        from (select e.deptno, e.ename, e.sal, avg(sal) over (partition by deptno) 부서평균
                from emp e)
                where sal > 부서평균
```

위의 SQL을 수행하려면 from절의 서브쿼리인 in line view를 써야합니다.

84. (11/20 시험 유형8) 점심시간문제. 부서번호, 이름, 월급, 순위를 출력하는데 순위가 부서번호별로 각각

월급이 높은 순서대로 순위를 부여해서 출력하고 순위가 1등인 사원들만 출력하시오!

```
select *
    from (select deptno, ename, sal, dense_rank() over (partition by deptno
                                                        order by sal desc) 순위)
    where 순위 = 1;
```

85. 사원 테이블에 loc 컬럼을 추가하고 해당 사원이 속한 부서위치로 값을 갱신하시오!

튜닝 전 :

```
alter table emp
    add loc varchar2(20);

update emp e
  set loc = (select loc
             from dept d
             where d.deptno = e.deptno);

select ename, loc
   from emp;
```

>>> 업데이트가 되었다.

설명 : emp테이블의 컬럼이 서브쿼리 안으로 들어가게 되면 서브쿼리부터 수행되는게 아니라 메인 쿼리(update)문부터 실행됩니다. update문을 수행하는데 제일 먼저 맨 위에 있는 KING의 부서위치를 갱신하기 위해서 KING의 부서번호 10번을 서브쿼리 안으로 넣어서 서브쿼리에서 10번 부서번호의 부서위치를 출력하고 KING의 부서위치를 NEW YORK으로 갱신합니다. 그 다음행의 BLAKE도 똑같은 방법으로 갱신하면서 EMP 테이블의 14명의 사원들의 데이터를 이렇게 갱신합니다. 그러면 문제가 update가 14번 수행됩니다.

튜닝 후: 다시 rollback하고 merge문으로 수행하시오!

```
merge into emp e
using dept d
on (e.deptno = d.deptno)
when matched then
update set e.loc = d.loc;
```

한번에 병합이 된다.

설명 : 튜닝전과는 다르게 튜닝후는 데이터를 한번에 갱신합니다.

86. 사원 테이블에 sal2라는 컬럼을 추가하시오!

```
alter table emp
add sal2 number(10);
```

87. 지금 추가한 sal2에 해당 사원의 월급으로 값을 갱신하시오.

```
update emp e
    set sal2 = sal;

@demo
alter table emp
add loc varchar2(20);
```

88. 아래의 복합뷰를 생성하고 뷰를 쿼리하시오!

```
create view emp_dept
as
select e.ename, e.loc as emp_loc, d.loc as dept_loc
    from emp e, dept d
    where e.deptno = d.deptno;
```

89. emp_dept뷰를 수정해서 emp_loc의 값을 dept_loc값으로 갱신하시오!

```
update emp_dept
    set emp_loc = dept_loc;
```

>>> 업데이트 안된다.

90. 문제 89번의 update문이 수행되게 설정하시오!

위의 복합뷰를 갱신하려면 dept테이블의 deptno에 primary key제약을 걸어주면 갱신될 수 있습니다.

만약 복합뷰를 갱신하려면 dept 테이블의 deptno에 primary key제약을 걸어주면 갱신 될 수 있습니다. 만약 dept테이블의 deptno에 pk가 없어서 20번이 여러개이고 30번이 여러 개이면 뷰 만들 때 생성했던 뷰 쿼리문의 결과가 믿을 수 없는 결과가 됩니다. 같은 20번인데 누구는 DALLAS이고 누구는

CHICAGO이고 이렇게 뒤죽박죽 되어버려서 뷰의 결과를 믿을수가없게 되면 그 믿을 수 없는 데이터를 어떻게 또 갱신할 수 있겠습니까?
그래서 pk제약을 걸어줍니다.

```
alter table dept
      add constraint dept_deptno_pk primary key(deptno);
```

>>> dept 테이블의 deptno에 중복된 데이터와 null값을 허용하지 않겠다라는 것을 완전히 책임지고 보증하겠다라고 합니다

```
update emp_dept
      set emp_loc = dept_loc;
```

이렇게 하면 merge문을 쓰는 것과 속도가 비슷하다.

91. 만약 dba가 view를 안 만들어줘서 view를 만들 수 없다면 어떻게 해야 하는가?

update <--- 서브쿼리 사용가능
set <--- 서브쿼리 사용가능
where <--- 서브쿼리 사용가능

```
update (select e.ename, e.loc as emp_loc, d.loc as dept_loc
            from emp e, dept d
            where e.deptno = d.deptno)
  set emp_loc = dept_loc;
```

한번에 빨리 갱신가능

설명 : 뷰를 만들어 주지 않는다면 update에 서브쿼리를 사용해서 뷰의 쿼리문을 직접 작성하고 set절에서 update하면 됩니다.

92. emp 테이블에 dname 컬럼을 추가하고 해당사원의 부서명으로 값을 갱신하시오!

```
alter table emp
      add dname varchar2(10);
```

튜닝 전 :

```
update (select e.ename, e.dname as emp_dname,
                        d.dname as dept_dname
                        from emp e, dept d
                        where e.deptno = d.deptno)
      set emp_dname = dept_dname;
```

튜닝 후 :

```
merge into emp e
      using dept d
      on (e.deptno = d.deptno)
      when matched then
      update set e.dname = d.dname;
```

삼성디스플레이에서 튜닝전 SQL로 40분 돌아가던게 튜닝 후 SQL로 1분걸리게 되었다.

93. emp_dept view와 salgrade 테이블을 서로 조인해서 이름과 월급과 부서위치와 급여등급(grade)을 출력하시오!

```
select v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
```

94. 문제 93번의 SQL의 실제 실행계획을 확인하시오!

```
select /*+ gather_plan_statistics leading(s v), use_nl(v) */ v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     14 |00:00:00.01 |      21 |       |       |          |
|*  1 |  HASH JOIN           |          |      1 |      1 |     14 |00:00:00.01 |      21 |  1506K|  1506K|  972K (0)|
|   2 |   MERGE JOIN         |          |      1 |      1 |     14 |00:00:00.01 |      14 |       |       |          |
|   3 |    SORT JOIN         |          |      1 |      5 |      5 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   4 |     TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       7 |       |       |          |
|*  5 |    FILTER            |          |      5 |        |     14 |00:00:00.01 |       7 |       |       |          |
```

```
|*  6 |     SORT JOIN          |          |      5 |     14 |     40 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   7 |      TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
|   8 |   TABLE ACCESS FULL   | DEPT     |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
```

설명 : 위의 힌트에서는 nested loop조인을 해라! 라고 했지만 실행계획에서는 nested loop 조인했다는 게 없습니다. 왜 힌트를 무시했느냐면 view를 해체해버려서 입니다. view가 해체되지 않았다면 실행계획에 view가 보입니다. 그래서 view를 해체하지 못하도록 힌트를 주어야 합니다. 그 힌트가 바로 no_merge입니다.

```
select /*+ gather_plan_statistics no_merge(v) leading(s v), use_nl(v) */ v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |     14 |00:00:00.01 |      21 |       |       |          |        (4)
|   1 |  MERGE JOIN            |           |      1 |      1 |     14 |00:00:00.01 |      21 |       |       |          |        (3)
|   2 |   SORT JOIN            |           |      1 |      5 |      5 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL   | SALGRADE  |      1 |      5 |      5 |00:00:00.01 |       7 |       |       |          |        (1)
|*  4 |   FILTER               |           |      5 |        |     14 |00:00:00.01 |      14 |       |       |          |
|*  5 |    SORT JOIN           |           |      5 |     14 |     40 |00:00:00.01 |      14 |  2048 |  2048 | 2048  (0)|
|   6 |     VIEW               | EMP_DEPT  |      1 |     14 |     14 |00:00:00.01 |      14 |       |       |          |        (2) 한묶음
|*  7 |      HASH JOIN         |           |      1 |     14 |     14 |00:00:00.01 |      14 |  1797K|  1797K| 1052K (0)|
|   8 |       TABLE ACCESS FULL| DEPT      |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|   9 |       TABLE ACCESS FULL| EMP       |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
------------------------------------------------------------------------------------------------------------------------- 실행순서
```

view가 보입니다.

힌트 : no_merge : view를 해체하지 말아라

merge : view를 해체해라~

실행계획 해석: 먼저 salgrade를 full scan하고 그리고 view를 스캔하면서 nested loop조인을 수행했습니다.

view안의 조인 문장의 테이블의 조인순서는 어떻게 변경해야 하나요?

```
select /*+ gather_plan_statistics no_merge(v) leading(s v), use_nl(v) */ v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
```

현재 view 안의조인의순서는 dept----> emp 순으로 읽고 있습니다.

그런데 emp-----> dept 순으로 할 수는 없나요?

할 수 있습니다.

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
           leading(v.e v.d) use_hash(v.d) */
v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |     14 |00:00:00.01 |      77 |       |       |          |
|   1 |  NESTED LOOPS          |           |      1 |      1 |     14 |00:00:00.01 |      77 |       |       |          |
|   2 |   TABLE ACCESS FULL    | SALGRADE  |      1 |      5 |      5 |00:00:00.01 |       7 |       |       |          |
|*  3 |   VIEW                 | EMP_DEPT  |      5 |      1 |     14 |00:00:00.01 |      70 |       |       |          |
|*  4 |    HASH JOIN           |           |      5 |     14 |     70 |00:00:00.01 |      70 |  1651K|  1651K|  994K (0)|
|   5 |     TABLE ACCESS FULL| EMP       |      5 |     14 |     70 |00:00:00.01 |      35 |       |       |          |
|   6 |     TABLE ACCESS FULL| DEPT      |      5 |      4 |     20 |00:00:00.01 |      35 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
```

순서가 바뀐 것을 볼 수 있다.

95. 위의 실행계획이 아래와 같이 나오게 하시오!

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |     14 |00:00:00.01 |     532 |
|   1 |  NESTED LOOPS          |           |      1 |      1 |     14 |00:00:00.01 |     532 |
|   2 |   TABLE ACCESS FULL    | SALGRADE  |      1 |      5 |      5 |00:00:00.01 |       7 |
|*  3 |   VIEW                 | EMP_DEPT  |      5 |      1 |     14 |00:00:00.01 |     525 |
|   4 |    NESTED LOOPS        |           |      5 |     14 |     70 |00:00:00.01 |     525 |
|   5 |     TABLE ACCESS FULL| EMP       |      5 |     14 |     70 |00:00:00.01 |      35 |
|*  6 |     TABLE ACCESS FULL| DEPT      |     70 |      1 |     70 |00:00:00.01 |     490 |
-------------------------------------------------------------------------------------------
```

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
           leading(v.e v.d) use_nl(v.d) */
v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

96. 위의 실행계획을 아래와 같이 출력되게 하시오!

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     14 |00:00:00.01 |     112 |       |       |          |
|   1 |  NESTED LOOPS       |          |      1 |      1 |     14 |00:00:00.01 |     112 |       |       |          |
|   2 |   VIEW              | EMP_DEPT |      1 |     14 |     14 |00:00:00.01 |      14 |       |       |          |
|*  3 |    HASH JOIN        |          |      1 |     14 |     14 |00:00:00.01 |      14 |  1651K|  1651K|  701K (0)|
|   4 |     TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       7 |       |       |          |
|   5 |     TABLE ACCESS FULL| DEPT     |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|*  6 |   TABLE ACCESS FULL | SALGRADE |      14 |      1 |     14 |00:00:00.01 |      98 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

```
select /*+ gather_plan_statistics no_merge(v) leading(v s) use_nl(s)
           leading(v.e v.d) use_hash(v.d) */
v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
```

97. 아래와 같이 실행계획이 나오게 하시오!

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     14 |00:00:00.01 |      77 |       |       |          |
|   1 |  NESTED LOOPS       |          |      1 |      1 |     14 |00:00:00.01 |      77 |       |       |          |
|   2 |   TABLE ACCESS FULL | SALGRADE |      1 |      5 |      5 |00:00:00.01 |       7 |       |       |          |
|*  3 |   VIEW              | EMP_DEPT |      5 |      1 |     14 |00:00:00.01 |      70 |       |       |          |
|*  4 |    HASH JOIN        |          |      5 |     14 |     70 |00:00:00.01 |      70 |  1797K|  1797K| 1055K (0)|
|   5 |     TABLE ACCESS FULL| DEPT     |      5 |      4 |     20 |00:00:00.01 |      35 |       |       |          |
|   6 |     TABLE ACCESS FULL| EMP      |      5 |     14 |     70 |00:00:00.01 |      35 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
           leading(v.d v.e) use_hash(v.e) */
      v.ename, v.sal, v.loc, s.grade
      from emp_dept v, salgrade s
      where v.sal between s.losal and s.hisal;
```

설명 : view를 사용하여 조인하는 어떤 SQL문장의 성능이 느리다면 위와 같은 방법으로 튜닝을 하면 효과적으로 튜닝을 할 수 있습니다.

98. (마지막문제) (SQL 알고리즘 문제24번) 1과 10사이의 숫자중에서 소수만 출력하시오!

```
select 숫자1 as 소수
   from (select level 숫자1
         from dual
         connect by level <= 10) lev1,
        (select level 숫자2
         from dual
         connect by level <=10) lev2
   where mod(lev1.숫자1, lev2.숫자2) = 0
   group by 숫자1
   having count(*) = 2;
```

99. 다음의 서브쿼리문을 조인으로 수행해서 같은 결과를 보시오!

DALLAS에서 근무하는 사원들의 이름과 월급을 출력하시오!

<보기>

```
select ename, sal
      from emp
      where deptno in ( select deptno
                        from dept
                        where loc = 'DALLAS' );
```

<답>

```
select e.ename, e.sal
      from emp e, dept d
      where e.deptno = d.deptno and d.loc = 'DALLAS';
```

100. 아래의 SQL의 실행계획이 조인으로 풀리게 하시오!

```
select ename, sal
      from emp
      where deptno in ( select deptno
                        from dept
                        where loc = 'DALLAS');
```

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ unnest */ deptno
```

```
                        from dept
                        where loc = 'DALLAS');
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      5 |00:00:00.01 |      12 |       |       |          |
|*  1 |  HASH JOIN SEMI    |      |      1 |      5 |      5 |00:00:00.01 |      12 |  1651K|  1651K|  985K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|*  3 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       6 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

설명 : unnest가 유리한 경우는 서브쿼리문의 실행계획이 filter가 나오면서 성능이 너무 느릴때 조인의 방법 중 가장 강력한 hash join으로 수행되게 하면서 성능을 높이고 싶을 때 유리합니다.

101. 아래의 실행계획이 조인으로 풀리지 않고 filter가 실행계획에 나오는 순수한 서브쿼리문으로 실행되게 하시오

(보기)

```
select ename, sal
      from emp
      where deptno in ( select deptno
                        from dept
                        where loc = 'DALLAS')
```

(답)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ no_unnest */ deptno
                        from dept
                        where loc = 'DALLAS');
-----------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      5 |00:00:00.01 |      27 |
|*  1 |  FILTER            |      |      1 |        |      5 |00:00:00.01 |      27 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |
|*  3 |   TABLE ACCESS FULL| DEPT |      3 |      1 |      1 |00:00:00.01 |      20 |
-----------------------------------------------------------------------------
```

설명 : no_unnest를 사용해서 순수하게 서브쿼리문으로 수행되었습니다. 그런데 버퍼의 개수가 27개로 아까 hash join semi로 수행되었을 때는 12개 였는데 약 2배가량 성능이 느려졌습니다.
위의 서브쿼리의 테이블 2개가 대용량 테이인 경우는 unnest를 써서 조인(해쉬조인)으로 수행되는게 유리하고 그렇지않고 테이블이 작다면 굳이 메모리를 사용하는 해쉬조인으로 유도하지 말고 서브쿼리문의 실행계획(filter)로 실행되는게 유리합니다.

102. 아래의 SQL의 실행계획을 확인하고 조인으로 변경되어서 수행되게 하시오!

(보기)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno = (select deptno
                        from dept
                        where deptno = 10 );
-----------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |      14 |              3
|*  1 |  TABLE ACCESS FULL | EMP  |      1 |      5 |      3 |00:00:00.01 |      14 |              2
|*  2 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       7 |              1
-----------------------------------------------------------------------------                 (실행계획)
```

설명 : 위의 SQL을 순수하게 서브쿼리문으로 수행한 SQL데 서브쿼리와 메인쿼리중에 어느것을 먼저 수행했냐면 서브쿼리문부터 수행하였습니다.

(답)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in (select /*+ unnest */deptno
                        from dept
                        where deptno = 10 );

-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |      11 |       |       |          |
|*  1 |  HASH JOIN SEMI    |      |      1 |      3 |      3 |00:00:00.01 |      11 |  1651K|  1651K|  724K (0)|
|*  2 |   TABLE ACCESS FULL| EMP  |      1 |      3 |      3 |00:00:00.01 |       6 |       |       |          |
|*  3 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       5 |       |       |          |
```

설명 : 위의 서브쿼리문과 메인쿼리문 사이의 연산자를 = 로 하면 unnest 힌트가 먹히지 않습니다. 왜냐하면 서브쿼리에서 메인쿼리로 한건만 리턴된다고하면 굳이 해쉬조인으로 풀지 않아도 되기 때문입니다. 그래서 만약 해쉬조인으로 수행되게하고 싶다면 연산자를 =에서 in으로 변경해줘야 합니다.

103. 아래의 SQL의 실행계획이 조인으로 풀리지 말고 서브쿼리문으로 수행되게 하는데 서브쿼리문부터 수행되게 하시오!

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno = ( select deptno
                       from dept
                       where deptno = 10);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |      14 |                  3
|*  1 |  TABLE ACCESS FULL | EMP  |      1 |      5 |      3 |00:00:00.01 |      14 |                  2
|*  2 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       7 |                  1
-------------------------------------------------------------------------------------                  실행순서
```

dept 읽고 emp 읽고 수행

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno = ( select /*+ no_unnest push_subq */ deptno
                       from dept
                       where deptno = 10);
```

이렇게 해도 위와 실행순서 동일

104. 문제 103번의 SQL의 실행계획이 이번에는 메인쿼리부터 수행되게 하시오! (순수하게 서브쿼리로 수행되면서 메인쿼리부터 수행되게 하시오!)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno = ( select /*+ no_unnest no_push_subq */ deptno
                       from dept
                       where deptno = 10);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |      14 |          4
|*  1 |  FILTER            |      |      1 |        |      3 |00:00:00.01 |      14 |          3
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |          1
|*  3 |   TABLE ACCESS FULL| DEPT |      1 |      1 |      1 |00:00:00.01 |       7 |          2
-------------------------------------------------------------------------------------      실행순서
```

push_subq와 no_push_subq는 no_unnest와 항상 함께 써야한다.

설명 : no_unnest와 no_push_subq는 서로 짝꿍 힌트입니다. no_unnest는 조인으로 풀지 말고 서브쿼리문으로 수행해라~ 라는 힌트이고 이 힌트를 먼저써줘야조인으로 풀지 않고 서브쿼리문으로 수행될 수 있기 때문에 no_push_subq힌트가 수행될 수 있었던 것 입니다. 대체로 push_subq 힌트가 서브쿼리문으로 수행되었을때는 더 유리한 힌트입니다. 왜냐하면 서브쿼리문부터 수행하면서 데이터를 검색해 메인쿼리로 넘겨주기만 하면 되기 때문입니다.
그런데 만약 메인쿼리부터 수행된다면 메인쿼리에 있는 부서번호중에 서브쿼리에 있는 부서번호를 찾기 위해 일일히 스캔하면서 찾는작업(filter)을 해야하기 때문에 성능이 대용량인 테이블의 경우는 성능이 많이 느려집니다.

105. 아래의 SQL이 순수하게 서브쿼리문으로 수행되게하고 서브쿼리부터 수행되게 하시오!

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select deptno
                        from dept );

select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ no_unnest push_subq */ deptno
                        from dept );
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      25 |          3
```

```
|*  1 |  TABLE ACCESS FULL | EMP  |      1 |      1 |     14 |00:00:00.01 |      25 |              2
|*  2 |   TABLE ACCESS FULL| DEPT |      3 |      1 |      3 |00:00:00.01 |      18 |              1
-------------------------------------------------------------------------------------                (실행순서)
```

설명 : no_unnest는 서브쿼리로 수행해라~ 라 것이고
push_subq는 서브쿼리부터 수행해라~~ 라는 것입니다.

106. 문제 105번의 SQL의 실행계획이 메인쿼리부터 수행되게 하세요~

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ no_unnest no_push_subq */ deptno
                              from dept );
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      25 |              4
|*  1 |  FILTER            |      |      1 |        |     14 |00:00:00.01 |      25 |              3
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       7 |              1
|*  3 |   TABLE ACCESS FULL| DEPT |      3 |      1 |      3 |00:00:00.01 |      18 |              2
-------------------------------------------------------------------------------------              실행순서
```

107. 문제 106번의 SQL이 조인으로 수행되게 하시오!
(조인 중에서 hash semi join으로 수행되게 하시오)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ unnest hash_sj*/ deptno
                              from dept );
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      11 |       |       |          |              4
|*  1 |  HASH JOIN SEMI    |      |      1 |     14 |     14 |00:00:00.01 |      11 |  1651K|  1651K|  958K (0)|              3
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |              1
|   3 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       5 |       |       |          |              2
-----------------------------------------------------------------------------------------------------------                (실행순서)
```

설명 : semi의 뜻은 절반이라는 뜻으로 조인을 했는데 완전한 조인을 한게 아니라 절반의 조인을 했습니다. 왜 완전한 조인을 못하고 절반의 조인을 했냐면 위의 SQL이 조인문장이 아니라 서브쿼리 문장이기 때문입니다.

108. 문제 107번의 실행계획이 dept가 메모리로 올라가게 하시오!

(dept가 hash table이 되게하시오!)

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ unnest hash_sj swap_join_inputs(dept) */ deptno
                              from dept );
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      12 |       |       |          |
|*  1 |  HASH JOIN RIGHT SEMI|     |      1 |     14 |     14 |00:00:00.01 |      12 |  2546K|  2546K| 1029K (0)|
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       6 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
-----------------------------------------------------------------------------------------------------------
```

실행계획이 바뀐 것을 볼 수 있다.

설명 : emp와 dept가 대용량 테이블이고 위와 같은 SQL이면 해쉬 세미조인으로 수행하되 작은 테이블이 메모리로 올라가게 힌트를준 위의 힌트가가장 모범적인 힌트입니다.

109. 문제 108번의 실행계획이 nested loop semi조인이 되게 하시오.

```
select /*+ gather_plan_statistics */ ename, sal
      from emp
      where deptno in ( select /*+ unnest nl_sj */ deptno
                              from dept d);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |      21 |
|   1 |  NESTED LOOPS SEMI |      |      1 |     14 |     14 |00:00:00.01 |      21 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |
|*  3 |   TABLE ACCESS FULL| DEPT |      3 |      4 |      3 |00:00:00.01 |      15 |
-------------------------------------------------------------------------------------
```

설명 : hash semi 조인때는 버퍼이 개수가 12ㅐ 였는데 nested loop semi조인 때는 버퍼의 개수가 14개로 늘어났습니다. 즉 hash semi조인을 사용하는게 훨씬 성능에 유리합니다.

110. 관리자인 사원들의 이름을 출력하시오!

(자기 밑에 직속부하가 한명이라도 있는 사원들)

```
select /* gather_plan_statistics */ ename
       from emp
       where empno in (select mgr
                       from emp);
```

```
-------------------------------------------------------------------------
| Id  | Operation          | Name | E-Rows |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |        |       |       |          |
|*  1 |  HASH JOIN SEMI    |      |      6 |  1856K|  1856K| 1429K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |     14 |       |       |          |
|*  3 |   TABLE ACCESS FULL| EMP  |     13 |       |       |          |
-------------------------------------------------------------------------
```

설명 : 실행계획을 보면 둘다 emp여서 메인쿼리부터 수행했는지 아니면 서브쿼리부터 수행했는지 확실히 알기가 어렵습니다.

111. 110번문제의 쿼리의 수행순서를 확실히 알 수 있도록 QB_NAME힌트를 써서 다시 실행하시오!

```
select /*+ gather_plan_statistics QB_NAME(main) */ ename
       from emp
       where empno in (select /*+ QB_NAME(sub) */ mgr
                       from emp);
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'advanced'));
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN SEMI    |      |     6 |    84 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   140 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP  |    13 |    52 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$526A7031
   2 - SEL$526A7031 / EMP@MAIN
   3 - SEL$526A7031 / EMP@SUB
```

설명 : format <= 'advanced' 를 사용하게 되면 좀 더 정보가 많은 실행계획을 확인 할 수 있습니다. QB_NAME 힌트를 사용하면 그 해당 쿼리의 이름을 지어주게 됩니다. 쿼리의 이름을 통해서 여기가 메인쿼리인지 서브쿼리인지 확인을할 수 있게 됩니다.

112. 점심시간 문제

주사위 한개를 288회 던져서 5이상의 눈이 나올 확률을 구하시오!

```
with d1 as
(select round(dbms_random.value(0.5, 6.5)) as 주사위
   from dual
   connect by level <= 288)
select count(주사위)/288
   from d1
   where 주사위 >= 5;
```

113. 아래의 쿼리의 실행계획이 해쉬 세미조인으로 수행되게하시오.

```
select /* gather_plan_statistics */ ename
       from emp
       where empno in (select /*+ unnest hash_sj */mgr
                       from emp);
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN SEMI    |      |     6 |    84 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   140 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP  |    13 |    52 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

114. 위의 SQL의 실행계획을 보면 메인쿼리부터 수행했는지 서브쿼리부터 수행했는지를 확인하기 위해 QB_NAME힌트를 써서 수행하시오.

```
select /*+ gather_plan_statistics QB_NAME(main) */ ename
       from emp
       where empno in (select /*+ QB_NAME(sub) unnest hash_sj  */ mgr
                       from emp);
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'advanced'));
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN SEMI    |      |     6 |    84 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   140 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP  |    13 |    52 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$0C6FB14C
   2 - SEL$0C6FB14C / EMP@SEL$1
   3 - SEL$0C6FB14C / EMP@SUB
```

설명 : 세미조인은 무조건 메인쿼리부터 수행됩니다. 그런데 swap_join_inputs을 쓰면 서브쿼리부터 수행되게 할 수 있습니다.

115. 위의 SQL이 subquery 부터 수행되어서 수행되는 hash semi join이 되게 하시오.

```
select /*+ gather_plan_statistics QB_NAME(main) */ ename
      from emp
      where empno in (select /*+ QB_NAME(sub) unnest hash_sj swap_join_inputs(emp@sub) */ mgr
                        from emp);
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'advanced'));
```

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN RIGHT SEMI|     |     6 |    84 |     6   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL | EMP  |    13 |    52 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL | EMP  |    14 |   140 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$526A7031
   2 - SEL$526A7031 / EMP@SUB
   3 - SEL$526A7031 / EMP@MAIN
```

설명 : 위와 같이 메인쿼리의 테이블과 서브쿼리의 테이블이 서로 같을 때 해쉬 세미조인을 하는경우 서브쿼리의 테이블부터 수행되게 하고싶다면 QB_NAME(sub) 힌트를 이용해서 쿼리블럭의 이름을 sub라고 주고 swap_join_inputs 괄호 안에 테이블 별칭으로 emp@sub를 사용하면 됩니다. 그러면 실행계획이 서브쿼리의 테이블부터 수행되면서 hash right semi join으로 수행됩니다.

116. 관리자가 아닌 사원들의 이름을 출력하시오!
(자기 밑에 직속 부하가 한명도 없는 사원들)

```
select ename
      from emp
      where empno not in (select mgr
                          from emp
                          where mgr is not null);
```

117. 문제 116번의 실행계획이 서브쿼리로 풀리지 않고 조인으로 풀리게 하시오!
(서브쿼리의 테이블과 메인쿼리의 테이블이 서로 대용량이면 해쉬 조인으로 풀리는게 훨씬 성능이 좋습니다.)

```
select /*+ gather_plan_statistics QB_NAME(main) */ename
      from emp
      where empno not in (select /*+ QB_NAME(sub) unnest hash_aj */ mgr
                          from emp
                          where mgr is not null);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN ANTI    |      |     8 |   112 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   140 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP  |    13 |    52 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$526A7031
   2 - SEL$526A7031 / EMP@MAIN
   3 - SEL$526A7031 / EMP@SUB
```

설명 : not in 을 사용한 서브쿼리 문장의 성능을 높이기 위해서는 hash anti조인을 사용하면 됩니다. 해쉬 안티 조인으로 수행되기 하기 위한 힌트는 unnest hash_aj입니다.
unnest 와 hash_aj는 짝꿍

118. 문제 117번의 해쉬조인 실행계획의 조인순서가 서브쿼리부터 수행되게 하시오.

```
select /*+ gather_plan_statistics QB_NAME(main) */ename
      from emp
      where empno not in (select /*+ QB_NAME(sub) unnest hash_aj
                                                    swap_join_inputs(emp@sub) */ mgr
                                 from emp
                                 where mgr is not null);
```

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |       |       |     6 (100)|          |
|*  1 |  HASH JOIN RIGHT ANTI|     |     8 |   112 |     6   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL | EMP  |    13 |    52 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL | EMP  |    14 |   140 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

```
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$526A7031
   2 - SEL$526A7031 / EMP@SUB
   3 - SEL$526A7031 / EMP@MAIN
```

그런데 이렇게 안나오면

```
select /*+ gather_plan_statistics QB_NAME(main) */ename
      from emp
      where empno not in (select /*+ QB_NAME(sub) unnest hash_aj
                                                    swap_join_inputs(emp@sub) */ mgr
                                 from emp
                                 where mgr is not null)
      and empno is not null;
```

이렇게 해준다.

설명 : not in을 사용한 서브쿼리문장의 성능을 높이기 위해서는 해쉬 안티 조인으로 수행하게 하면 되는데 서브쿼리부터 수행되게 하려면 swap_join_inputs를 써서 hash right anti join으로 수행되게 하면 됩니다.

119. 아래의 SQL을 튜닝하시오!

튜닝 전 : select deptno, sum(sal)

```
          from emp
          group by deptno
        union all
        select null as deptno, sum(sal)
          from emp
          order by deptno asc;
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      4 |00:00:00.01 |      12 |       |       |          |
|   1 |  UNION-ALL          |      |      1 |        |      4 |00:00:00.01 |      12 |       |       |          |
|   2 |   HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       6 |  1200K|  1200K| 1253K (0)|
|   3 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   4 |   SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.01 |       6 |       |       |          |
|   5 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
-----------------------------------------------------------------------------------------------------------
```

튜닝 후 : select /*+ gather_plan_statistics */ deptno, sum(sal)

```
          from emp
          group by rollup(deptno);
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      4 |00:00:00.01 |       6 |       |       |          |
|   1 |  SORT GROUP BY ROLLUP|     |      1 |      3 |      4 |00:00:00.01 |       6 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL | EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
-----------------------------------------------------------------------------------------------------------
```

설명 : 버퍼의 개수가 튜닝전에는 12개였는데 튜닝후에는 6개로 두배 빨라졌습니다.

120. 아래의 SQL을 튜닝하시오!

튜닝 전 : select /*+ gather_plan_statistics */ deptno, null as job, sum(sal)

```
  from emp
  group by deptno
union all
select null as deptno, job, sum(sal)
  from emp
  group by job
  order by deptno asc, job asc;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |      1 |        |      8 |00:00:00.01 |      12 |       |       |          |
|   1 |  SORT ORDER BY       |      |      1 |      8 |      8 |00:00:00.01 |      12 |  2048 |  2048 | 2048  (0)|
|   2 |   UNION-ALL          |      |      1 |        |      8 |00:00:00.01 |      12 |       |       |          |
|   3 |    HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       6 |  1200K|  1200K| 1255K (0)|
|   4 |     TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   5 |    HASH GROUP BY     |      |      1 |      5 |      5 |00:00:00.01 |       6 |  1116K|  1116K|  745K (0)|
|   6 |     TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
-------------------------------------------------------------------------------------------------------
```

튜닝 후 : select /*+ gather_plan_statistics */ deptno, job, sum(sal)
from emp
group by grouping sets((deptno) , (job))
order by 1 asc, 2 asc;

```
-------------------------------------------------------------------------------------------------------------------------------------------
-
| Id  | Operation                           | Name                    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-
Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
-
|   0 | SELECT STATEMENT                    |                         |      1 |        |      8 |00:00:00.01 |       9 |       |       |          |
|   1 |  TEMP TABLE TRANSFORMATION          |                         |      1 |        |      8 |00:00:00.01 |       9 |       |       |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D660C_50AB15 |      1 |        |      0 |00:00:00.01 |       8 |  1024
|  1024 |          |
|   3 |    TABLE ACCESS FULL                | EMP                     |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   4 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D660D_50AB15 |      1 |        |      0 |00:00:00.01 |       0 |  1024
|  1024 |          |
|   5 |    HASH GROUP BY                    |                         |      1 |      5 |      5 |00:00:00.01 |       0 |  1116K|  1116K|  745K (0)|
|   6 |     TABLE ACCESS FULL               | SYS_TEMP_0FD9D660C_50AB15 |      1 |     14 |     14 |00:00:00.01 |       0 |       |       |
|
|   7 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D660D_50AB15 |      1 |        |      0 |00:00:00.01 |       0 |  1024
|  1024 |          |
|   8 |    HASH GROUP BY                    |                         |      1 |      3 |      3 |00:00:00.01 |       0 |  1200K|  1200K| 1254K (0)|
|   9 |     TABLE ACCESS FULL               | SYS_TEMP_0FD9D660C_50AB15 |      1 |     14 |     14 |00:00:00.01 |       0 |       |       |
|
|  10 |   SORT ORDER BY                     |                         |      1 |     11 |      8 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|  11 |    VIEW                             |                         |      1 |      5 |      8 |00:00:00.01 |       0 |       |       |          |
|  12 |     TABLE ACCESS FULL               | SYS_TEMP_0FD9D660D_50AB15 |      1 |      5 |      8 |00:00:00.01 |       0 |       |       |
|
-------------------------------------------------------------------------------------------------------------------------------------------
```

버퍼의 개수가 8개로 줄었다.

121. 아래의 SQL을 튜닝하시오.

튜닝 전: select /*+ gather_plan_statistics */ deptno, null as job, sum(sal)

```
  from emp
  group by deptno
union all
select null as deptno, job, sum(sal)
  from emp
  group by job
  order by deptno asc, job asc;
union all
select null as deptno, null as job, sum(sal)
  from emp
  order by deptno asc, job asc;
```

튜닝 후 :

```
select /*+ gather_plan_statistics */ deptno, job, sum(sal)
                  from emp
               group by grouping sets((deptno) , (job), ())
order by deptno asc, job asc;
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
-
| Id  | Operation                           | Name                    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-
Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
-
|   0 | SELECT STATEMENT                    |                         |      1 |        |      9 |00:00:00.01 |       8 |       |       |          |
|   1 |  TEMP TABLE TRANSFORMATION          |                         |      1 |        |      9 |00:00:00.01 |       8 |       |       |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6610_50AB15 |      1 |        |      0 |00:00:00.01 |       7 |  1024
|  1024 |          |
|   3 |    TABLE ACCESS FULL                | EMP                     |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   4 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6611_50AB15 |      1 |        |      0 |00:00:00.01 |       0 |  1024
```

```
| 1024 |          |
|  5 |   HASH GROUP BY                    |                          |      1 |      5 |      5 |00:00:00.01 |       0 |  1116K|  1116K|  747K (0)|
|  6 |    TABLE ACCESS FULL               | SYS_TEMP_0FD9D6610_50AB15 |      1 |     14 |     14 |00:00:00.01 |       0 |       |       |          |
|
|  7 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6611_50AB15 |      1 |        |      0 |00:00:00.01 |       0 |  1024
| 1024 |          |
|  8 |   SORT GROUP BY ROLLUP             |                          |      1 |      3 |      4 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|  9 |    TABLE ACCESS FULL               | SYS_TEMP_0FD9D6610_50AB15 |      1 |     14 |     14 |00:00:00.01 |       0 |       |       |          |
|
| 10 |   SORT ORDER BY                    |                          |      1 |     11 |      9 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
| 11 |    VIEW                            |                          |      1 |      5 |      9 |00:00:00.01 |       0 |       |       |          |
| 12 |     TABLE ACCESS FULL              | SYS_TEMP_0FD9D6611_50AB15 |      1 |      5 |      9 |00:00:00.01 |       0 |       |       |          |
|
-------------------------------------------------------------------------------------------------------------------------------------------
```

122. 아래의 SQL을 튜닝하시오!

튜닝 전: select empno, ename, sal, (select sum(sal)
from emp s
where s.empno < m.empno) 누적치
from emp m
order by sal asc;

튜닝 후 : select /*+ gather_plan_statistics */ empno, ename, sal,
sum(sal) over (order by empno asc) 누적치
from emp;

설명 : 튜닝전은 emp 테이블을 2번 엑세스 했으나 튜닝후는 한번만 엑세스 했습니다.

123. 아래의 SQL을 튜닝하시오!

부서번호별로 각각 월급을 누적 시키고 있습니다.

튜닝 전 :

select deptno, empno, ename, sal, (select sum(sal)
from emp s
where s.empno < m.empno
and s.deptno = m.deptno) 누적치
from emp m
order by sal asc;

튜닝 후 :

```
select /*+ gather_plan_statistics */ deptno, empno, ename, sal,
        sum(sal) over (partition by deptno
        order by empno asc) 누적치
from emp;
```

124. 아래의 SQL을 튜낭하시오! (SQLP 시험 주관식 3문제중 한문제)

create index emp_ename on emp(ename);

튜닝 전 : select ename, sal, job
from emp
where ename like '%EN%' or ename like '%IN%';

또는

```
Select /*+ gather_plan_statistics */ ename, age
From emp12
Where regexp_like(ename,'정|준');
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      3 |00:00:00.01 |       6 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      3 |00:00:00.01 |       6 |
-------------------------------------------------------------------------------
```

설명 : ename에 아무리 인덱스가 있다 하더라도 like 연산자 사용할 때 와일드 카드(%)가 앞에 나오면 인덱스를 엑세스 하지 못하고 full table scan합니다. 버퍼의 개수가 6개 나왔습니다.

튜닝 후 :

I. **먼저 이름에 EN또는 IN이 포함되어 있는 사원의 ROWID를 emp_ename의 인덱스를 통해서 알아냅니다. 빠르게!!! 알아냅니다.**

```
select /*+ gather_plan_statistics index_ffs(emp emp_sal) */ rowid
        from emp
        where ename like '%IN%' or ename like '%EN%';
```

>>> 그냥하면 index full scan 하기 때문에 not null제약을 걸어줌

```
alter table emp
      modify ename constraint emp_ename_nn not null;
```

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |      1 |        |      3 |00:00:00.01 |       4 |
|*  1 |  INDEX FAST FULL SCAN| EMP_ENAME |      1 |      3 |      3 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------
```

II. 알아낸 rowid를 통해서 테이블에서 해당 데이터를 검색하는데 nested loop조인으로 검색합니다.

```
select /*+ leading (v e) use_nl(e) */ e.ename, e.sal, e.job
      from emp e,
      (select /*+ gather_plan_statistics index_ffs(emp emp_ename) */ rowid as rn
            from emp
            where ename like '%IN%' or ename like '%EN%') v
      where e.rowid = v.rn;
```

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      3 |00:00:00.01 |       7 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      3 |      3 |00:00:00.01 |       7 |
-------------------------------------------------------------------------------
```

설명 : from절 서브쿼리인 인라인뷰에서 emp_ename 인덱스를 빠르게 스캔해서 rowid 3개를 알아낸 다음 emp 테이블과 조인했습니다. 실행계획을 보면 full table emp로 풀리면서 from절의 서브쿼리인 in line view를 해채해버렸습니다.

인라인뷰를 해체하지 못하도록 no_merge 힌트를 써서 해보겠습니다.

```
select /*+ leading (v e) use_nl(e) no_merge(v) */ e.ename, e.sal, e.job
      from emp e,
      (select /*+ gather_plan_statistics index_ffs(emp emp_ename) */ rowid as rn
            from emp
            where ename like '%IN%' or ename like '%EN%') v
      where e.rowid = v.rn;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      3 |00:00:00.01 |       5 |
|   1 |  NESTED LOOPS                |           |      1 |      3 |      3 |00:00:00.01 |       5 |
|   2 |   VIEW                       |           |      1 |      3 |      3 |00:00:00.01 |       4 |
|*  3 |    INDEX FAST FULL SCAN      | EMP_ENAME |      1 |      3 |      3 |00:00:00.01 |       4 |
|   4 |   TABLE ACCESS BY USER ROWID| EMP       |      3 |      1 |      3 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------
```

설명 : 튜닝전보다 버퍼 개수가 더 줄었습니다.

125. 우리반 테이블에서 성과 이름 전부 다해서 '정'자 '준' 자를 포함하고있는 학생들의 이름과 나이를 출력하시오!

```
create index emp12_ename on emp12(ename);

select /*+ gather_plan_statistics no_merge(v) leading (v e) use_nl(e) */ e.ename, e.age
from emp12 e, (select /*+ index_ffs(emp12 emp12_ename) */ rowid as rn
                  from emp12
                  where ename like '%정%' or ename like '%준%') v
where e.rowid = v.rn;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |      9 |00:00:00.01 |       5 |
|   1 |  NESTED LOOPS                |             |      1 |      3 |      9 |00:00:00.01 |       5 |
|   2 |   VIEW                       |             |      1 |      3 |      9 |00:00:00.01 |       4 |
|*  3 |    INDEX FAST FULL SCAN      | EMP12_ENAME |      1 |      3 |      9 |00:00:00.01 |       4 |
|   4 |   TABLE ACCESS BY USER ROWID| EMP12       |      9 |      1 |      9 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------
```

126. (오늘의 마지막 문제)

숫자를 물어보게 하고 숫자를 입력하면 해당숫자가 정수인지 소수인지 출력되게 하시오!

숫자를 입력하세요~ 2

정수 입니다.

숫자를 입력하세요~ 7
소수입니다.

주요 SQL

2020년 11월 20일 금요일 오전 10:18

view 공부하기

merge 공부하기

결과를 보는 SQL

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'advanced'));
```

해시 조인때 조인 순서를 바꾸는 함수:

swap_join_inputs